UNIVERSIDADE FEDERAL DE SANTA CATARINA
DEPARTAMENTO DE ENGENHARIA MECÂNICA

Victor Carreto

ESTUDO DE MECANISMOS AUTO-ALINHÁVEIS USANDO ANÁLISE DE DEPENDÊNCIAS ESTÁTICAS E CINEMÁTICAS.

Florianópolis

2010

Victor Carreto

ESTUDO DE MECANISMOS AUTO-ALINHÁVEIS USANDO ANÁLISE DE DEPENDÊNCIAS ESTÁTICAS E CINEMÁTICAS.

Dissertação submetida ao Programa de Pós-Graduação em Engenharia Mecânica para a obtenção do Grau de Mestre em Engenharia Mecânica.
Orientador: Prof. Daniel Martins, Dr. Eng.
Coorientador: Prof. Henrique Simas, Dr. Eng.

Florianópolis

2010

Catalogação na fonte elaborada pela biblioteca da
Universidade Federal de Santa Catarina

A ficha catalográfica é confeccionada pela Biblioteca Central.

Tamanho: 7cm x 12 cm

Fonte: Times New Roman 9,5

Maiores informações em:
http://www.bu.ufsc.br/design/Catalogacao.html

Victor Carreto

ESTUDO DE MECANISMOS AUTO-ALINHÁVEIS USANDO ANÁLISE DE DEPENDÊNCIAS ESTÁTICAS E CINEMÁTICAS.

Esta dissertação foi julgada adequada para a obtenção do título de **MESTRE EM ENGENHARIA MECÂNICA** e aprovada em sua forma final a pelo Programa de Pós-Graduação em Engenharia Mecânica.

Florianópolis, 21 de Dezembro 2010.

Prof. Daniel Martins, Dr. Eng.
Orientador

Prof. Henrique simas, Dr. Eng.
Coorientador

Prof. Eduardo Alberto Fancello, D.Sc.
Coordenador do Programa de Pós-Graduação em Engenharia Mecânica

BANCA EXAMINADORA:

Prof. Lauro Cesar Nicolazzi Dr. Eng.
Presidente

Prof. Altamir Dias, Dr. Sc.

Prof. Rodrigo de Souza Vieira, Dr. Eng.

Este trabalho é dedicado a minha família, especialmente a minha querida esposa Lilian por seu apoio incondicional e a meu filho Pedro, a grande motivação da minha vida.

AGRADECIMENTOS

Ao Professor Dr. Eng. Daniel Martins, pela orientação, confiança e sabedoria que me guiou nesta trajetória científica.
Aos professores que fizeram parte desta etapa acadêmica: Prof. Dr. Sc. Altamir Dias, Prof. Dr. Eng. André Ogliari e Prof. Dr. Eng. Henrique Simas.
Aos colegas do Laboratório de Robótica Professor Raul Gunther, em especial a João Victor e Frank Ajata pela amizade.
A minha amada esposa Lilian pela parceria incondicional, e meu filho Pedro que aguentou firme na creche para que este trabalho ficase completo. Aos meus pais Arq. Victor Carreto Valadez e Marcela Pavon de Carreto que sempre me apoiaram a continuar estudando e meus sogros Prof. Dr. Reinaldo Matias Fleuri e Dionisia Jurkevicz Fleuri pelo incentivo e o carinho com que sempre falam.
À Universidade Federal de Santa Catarina em específico ao Programa de Pós-Graduação em Engenharia Mêcanica, por me dar a oportunidade de continuar estudando e à CAPES pelo suporte financeiro.

RESUMO

Esta pesquisa foi desenvolvida para projetar mecanismos com características de auto-alinhamento, baseado em análises de dependências lineares. O auto-alinhamento refere-se à ação de projetar mecanismos com graus de liberdade (fornecidos pelas juntas) que facilitam a montagem. Faz-se uso da Teoria de Helicóides para analisar as dependências no mecanismo. Além disso apresentam-se as técnicas de projeto MinCD (Minimum Constraint Design), RedCD (Redundant Constraint Design) e as características básicas do Auto-Alinhamento. Uma das vantagens de se projetar mecanismos com auto-alinhamento é fornecer ao projetista a possibilidade de montar mecanismos e estruturas com mínimos problemas de tolerância e de esforços internos. O projeto de mecanismos, evita a geração de mobilidades perigosas. As técnicas de projeto MinCD e RedCD são explicadas e exemplificadas a fim de mostrar quais são os benefícios de se conhecer as restrições envolvidas no projeto, assim como a exposição do algoritmo desenvolvido para realizar a análise. O algoritmo permite sistematicamente identificar quais são as dependências lineares no mecanismo a fim de obter uma relação das dependências lineares e as restrições redundantes.

Palavras-chave: auto-alinhamento; restrições redundantes; dependência linear; mecanismos; MinCD; RedCD.

ABSTRACT

This research was developed to design mechanisms with self-aligning characteristics, based on a linear dependency analysis. Self-aligning is related to the action of designing mechanisms with degrees of freedom (allowed by the joints) that improve the assembly process. Screw Theory is used in the determination of dependencies in a mechanism. The text presents the design techniques using Minimum Constraint Design (MinCD), Redundant Constraint Design (RedCD), and the basic characteristics of self-aligning. One of the advantages of design mechanism with self-aligning is that allows the assembly of mechanisms with small tolerance variations and prevents from internal stress. This design technique avoids the generation of dangerous mobility's or underconstraints. MinCD and RedCD are explained to expose the advantage of knowing the constraints in a mechanism and the benefits of each design technique. The research proposes an algorithm developed to systematically analyze and identify linear dependencies to relate such result with the redundant restrictions in a mechanism.

Keywords: Self-aligning; Overconstraint; Linear dependencies; Mechanisms; MinCD; RedCD.

LIST OF FIGURES

Figure 1 Example of an overconstraint mechanism, (SKAKOON, 2009). 27
Figure 2 Example of a self aligning wheel caster, (KAMM, 1990) Fig. 3.27. 28
Figure 3 Illustration of mechanisms and their kinematic chain representation, a) (TSAI, 1999) Fig. 1.2, b) (TSAI, 2001) Fig. 1.6 and c) (BROWN, 1995) Fig. 169. 33
Figure 4 a) The three dimensional coordinate system with origin at $\mathcal{O}_{xyz}$ and the six freedoms allowed, in b) the two dimensional coordinate system with origin at $\mathcal{O}_{xy}$ and the three freedoms. 34
Figure 5 Higher pairs: a) point contact, b) line contact and c) curve contact, (HUNT, 1978). 36
Figure 6 Lower pairs: a) Spherical (S-pair), b) Planar pair (E-pair), c) Cylindrical pair (C-pair), d) Rotative pair (R-pair), e) Prismatic pair (P-pair) and f) Screw pair (H-pair), (HUNT, 1978). 38
Figure 7 Mechanisms in Figure 3 and the corresponding structural graph representation. 42
Figure 8 Exact Constraint Coupling (Kelvin's Coupling), (HALE; SLOCUM, 2001). 46
Figure 9 Examples of Red CD, they allow more stability to the design, (KAMM, 1990). 47
Figure 10 Unconstrained shaft, supported by two walls. 47
Figure 11 Four bar mechanism with serious geometric errors that impede the assembly, (SZYDLOWSKI, 2000). 48
Figure 12 Exagerated image of a) Overconstrainted bolted foot and b) self-aligning bolted foot with spherical washers, (KAMM, 1990). 49
Figure 13 active self-aligning mechanism, representation of a roller assembly from a garage door, (DOWNEY; PARKINSON; CHASE, 2003). 50
Figure 14 Table Method applied to a four bar mechanism with: *a*) four rotative joints, *b*) one spherical and three rotative joints, and *c*) two spherical and two rotative joints. 54
Figure 15 Graph representation of an S-pair, using virtual nodes. . 57
Figure 16 Graph representation for the actions in an S-pair. 60
Figure 17 Closed Passive coupling networks, (DAVIES, 2000). 63

Figure 18 a) Mechanism with cross paths, the fixed pairing element is at the bottom, in b) the fixed element changed is the left pairing element, eliminating the cross paths. 64
Figure 19 Graph representation for the Path Method: a) two pairing elements connected by a kinematic pair, b) union operation for two joints in series, c) intersection operation for two joints in parallel. 64
Figure 20 Diagram representing the first step. 70
Figure 21 Diagram representing the second step. 70
Figure 22 Diagram representing the third step. 71
Figure 23 Case I: Freedoms and constraints for a slider-crank mechanism. 72
Figure 24 Case II: Freedoms and constraints for a slider-crank mechanism. 75
Figure 25 Wrench, geometric representation, taken from (CAZANGI, 2008) Fig 3.8 and 3.9 96
Figure 26 Twist geometric representation, taken by (CAMPOS, 2004) Fig. 10. 98
Figure 27 Bipartite graph of the matrix **A** shown in matrix C.1 . . 102
Figure 28 Bipartite graph of maximum matching **A** shown in matrix C.1 102

LIST OF TABLES

Table 1 Gogu's terminology for the parameter λ, (GOGU, 2008).. 35
Table 2 Summarized *doc* and *dof* of kinematic pairs..... 38
Table 3 Rules for Exact CD, taken from (HAMMOND, 2004), p.20 45

LIST OF ABBREVIATIONS

IFToMM International Federation for the Promotion of Mechanism and Machine Science 25
CD Constraint Design 26
dof Degrees of freedom 34
doc Degrees of constraint 34
GNU GNU's Not UNIX! 40

LIST OF SYMBOLS

$\mathcal{O}_{xyz}$	Origin of a three dimensional space	32
f	Freedom	34
c	Constraint	34
λ	System order	34
f_{ij}	Relative freedom between the bodies i and j	36
c_{ij}	Relative constraint between the bodies i and j	36
T	Translation	37
R	Rotation	37
M	Mobility	39
j	number of kinematic pairs	39
V	vertices	40
E	Edges	40
G	Graph	40
$[A]$	Adjacency matrix	40
$[I]$	Incidence matrix	41
G_c	Coupling Graph	41
$[B]$	Circuit-f matrix	42
$[Q]$	Cutset-f matrix	42
\$	Screw	43
$\vec{v}$	Free vector	43
$\vec{u}$	Line vector	43
$\vec{S}$	Direction vector of the screw axis	43
$\vec{S_0}$	Position vector from the origin of the coordinate system to the axis of the screw	43
h	Pitch	43
$\hat{\$}$	Unitary Screw	44
l	number of f-circuits	51
G_M	Motion graph	56
F	Gross degree of freedom of a coupling network	56
$[M_D]$	Motion matrix	57
$[\hat{M}_D]$	Unitary motion matrix	57
$\{\Phi\}$	Magnitude of the Motion system	57

$[M_N]$	Motion network matrix	58
$[\hat{M}_N]$	Unitary network matrix	58
m	Rank of $[M_N]$	59
F_N	Net degrees of freedom	59
C_N	Net degrees of constraint	59
G_A	Action graph	60
C	Gross degree of constraint of a coupling network	60
k	Number of f-cuts	60
$[A_D]$	Action matrix	61
$[\hat{A}_D]$	Unitary action matrix	61
$\{\vec{\Psi}\}$	Magnitude of the action sytem	61
$[A_N]$	Network action matrix	61
$[\hat{A}_N]$	Unitary etwork action matrix	61
a	Rank of $[A_N]$	62
$\$^M$	Twist	89
$\vec{v}$	Linear velocity vector of a body	89
$\vec{\omega}$	Angular velocity vector of a body	89
ϕ	Magnitude of a twist	89
$\vec{R}_R$	Force vector in a body	95
$\vec{T}_R$	Moment vector in a body	95
$\$^A$	Wrench	95
ψ	Magnitude of a wrench	95

CONTENTS

1 INTRODUCTION 25
1.1 INTRODUCTION TO KINEMATICS OF A MECHANISM . 25
1.2 CONSTRAINT DESIGN 26
1.3 DISSERTATION OBJECTIVES 27
1.4 CHAPTER OVERVIEW 29
2 LITERATURE REVIEW 31
2.1 MECHANISM AND KINEMATIC CHAIN 31
2.1.1 Freedom and Constraint 32
2.1.2 Joints: Kinematic Pairs 36
2.1.3 Mobility Formulae 39
2.2 REPRESENTATION OF A MECHANISM 39
2.2.1 Graph Theory Applied to Mechanisms 40
2.2.2 Screw Theory 42
2.3 CONSTRAINT DESIGN (CD) 44
2.3.1 Minimum CD / Exact CD 45
2.3.2 Redundant CD / Over-constraint 45
2.3.3 Under-constraint 47
2.3.4 Self-aligning 48
3 METHODOLOGY: LITERATURE REVIEW 51
3.1 RESHETOV: TABLE METHOD 51
3.2 DAVIES METHOD 56
3.2.1 Instantaneous Kinematics 56
3.2.2 Statics 60
3.3 PATH METHOD 62
3.4 CHAPTER CONCLUSIONS 66
4 ANALYSIS 67
4.1 LINEAR INDEPENDENCE AND RANK 67
4.2 ALGORITHM 69
5 CONCLUSIONS 79
References 83
APPENDIX A – Mozzi Axis 89
APPENDIX B – Poinsot Axis 95
APPENDIX C – Dulmage and Mendelsohn Decomposition 101
APPENDIX D – Action matrices for Case I and Case II. 105
APPENDIX E – Twist - $_m.m 115
APPENDIX F – Wrench - $_a.m 119
APPENDIX G – davies.m 123

APPENDIX H – state.m 127
APPENDIX I – kinematic.m 131
APPENDIX J – static.m 139
APPENDIX K – find_col_zeros.m 147
APPENDIX L – enc.m 153
APPENDIX M – ex_1.m 159

1 INTRODUCTION

The objective of the research is to propose an algorithm to search for linearly dependent motions in a mechanism. This dissertation is related to the science of mechanisms, it presents a review on the subject by explaining theories and definitions that will help to understand the mechanisms and how they can be analyzed. The analysis examines mechanisms based on their structure and their geometrical properties so they can be classified by their state of constraint. The dissertation, also presents a review of literature that will help to analyze mechanisms in order to propose self-aligning qualities that can improve the design. This research developed a set of algorithms that identify, in a motion system and in an action system, which motions or actions are independent and which depend on others to work. Such relationships are based on the motions allowed by the joints and forces acting on the mechanical system. Therefore, this dissertation proposes a computer algorithm for kinematic analysis based on the review of literature presented in this investigation.

This chapter is divided into four sections that introduce ideas and definitions which are related to the science of mechanisms. In section 1.1, an introduction to kinematics of a mechanism, the study of motions, is presented to understand what kinematic analysis is. In section 1.2, the constraint conditions of a mechanism and the self-aligning constraint characteristic are briefly introduced. In section 1.3, the objectives of this dissertation are described, as is their relevance. Section 1.4 gives an outline of the chapters that constitute this dissertation.

1.1 INTRODUCTION TO KINEMATICS OF A MECHANISM

According to the International Federation for the Promotion of Mechanism and Machine Science (IFToMM), a *mechanism* is a: *"System of bodies designed to convert motions of, and forces on, one or several bodies into constrained motions of, and forces on, other bodies"*(IONESCU, 2003). In (HUNT, 1978) *Kinematics* is defined as: *"The branch of dynamics that deals with motion on its own in isolation from the forces associated with motion"*. For that reason, *kinematics of a mechanism* studies the relative motion between the connected bodies with reference to a fixed one, without considering the force that causes the motion or the friction of the system. Kinematic design is also be

related to theoretically constrained motion, using the minimum points of contact (HAMMOND, 2004). The kinematics of a mechanism can be divided into *kinematic synthesis* and *kinematic analysis* as is stated in (TSAI, 2001) and (GOGU, 2008). The kinematic synthesis is used in the design of new mechanisms, based on the desired motions that satisfy project objectives. This type of kinematic design is divided in to three phases, *type synthesis*, *number synthesis* and *dimensional synthesis*. Type synthesis is used to select the type of mechanism. During the conceptual design phase, all the possibilities are identified to select the type of mechanism that can best satisfy the objectives. For example, it can be a linkage, a cam, a gear mechanism or another system. Number synthesis deals with the number of bodies and joints that will define the motion of the mechanism; furthermore it enumerates possible kinematic structures that achieve the desired motion. Dimensional synthesis works to determine the dimensions and proportions of the bodies that will compose the mechanism. A kinematic analysis, the converse of kinematic synthesis, is an inspection realized on a mechanism where all the constraints are already predefined. The goal of a kinematic analysis is to examine and improve the design of a mechanism. In sections 3.2.1 and 3.2.2 a methodology based on (DAVIES, 2000) and (DAVIES, 2006) for kinematic analysis using instantaneous kinematics and

1.2 CONSTRAINT DESIGN

Constraint Design (CD) is introduced as a way to determine motion in rigid bodies and structures by constraining motion components. In both bodies and structures, motion can take place in exactly six motion components referred to a Cartesian coordinate system. Motion can be designed to occur in certain directions specified by the objectives of a project, so the right constraints must be chosen. Exact and Minimum CD are designing options where exactly six degrees of freedom are constrained (SKAKOON, 2009). Redundant CD or overconstraint is a possibility of design where more than six constraints define the state of restriction (KAMM, 1990). Statically indeterminated forces present advantages and disadvantages both mentioned in section 2.3.2. The underconstraint condition exists when one or more motion components are left without restriction, meaning that bodies are free to move. Unconstrained motions lead to a poor and unstable design, it can even be dangerous since the parts are not fully determined (SMITH, 2001). To consider self-aligning in a design will allow motion capable of

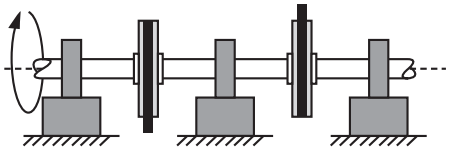


Figure 1 – Example of an overconstraint mechanism, (SKAKOON, 2009).

eliminating redundant constraints where undesired, (RESHETOV, 1979) proposes a method to eliminate overconstrainment.

In section 2.3 the constraint conditions are further explained for a better understanding of the properties of mechanisms and kinematic connections under each state of constraint. The intention of the kinematic analysis, introduced in section 1.1, is to define the state of constraint of a given mechanism in the four possible constraint conditions. The relevance of revealing the state of constraint of a mechanism helps the designer to improve the mechanical project by proposing changes to the design. To understand the importance of CD in mechanisms, figure 1 illustrates an overconstraint mechanism, a rotative shaft supported by three bearings. The mechanism is determining an axis of the shaft with three constraints, when in reality only two points are needed to define an axis. The assembly of such a mechanism will require extreme precision aligning and fixing the bearings, yet still it will not work correctly. Since an axis is determined by two points in the space, the static forces and moments of the mechanism will then be undetermined (BEDFORD; FOWLER, 2008).

1.3 DISSERTATION OBJECTIVES

As expressed in the beginning of the chapter, the objective of this dissertation is: first, present a review of literature related to mechanisms and their design constraints; second, propose an algorithm for kinematic analysis that identifies the state of constraint of a given mechanism in order to suggest changes in the design that can achieve self-aligning characteristics described in section 2.3.4.

The relevance of this research in the CD literature is that it presents a definition for self-aligning and its characteristics, based on (DAVIES, 1970), (DOWNEY; PARKINSON; CHASE, 2003), and (RESHETOV, 1979). It has the intention to illustrate and explain in a clear

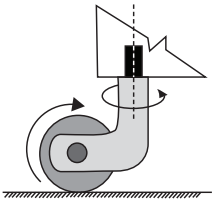


Figure 2 – Example of a self aligning wheel caster, (KAMM, 1990) Fig. 3.27.

manner the concepts that are involved with mechanisms, their representation and their analysis. A way to improve a mechanical design is by performing a kinematic analysis and determining the state of constraint of the assembly. The issues bestowed to the mechanical assemblies are: the under-constraint state, in which because of the undetermined mobility the mechanism is considered as dangerous and potentially harmful; and the over-constrained state, which can also be considered as an issue, since with redundant constraints the material can suffer internal stress or deformation as mentioned above, although sometimes deformation can be desired to permanently fix two elements. Authors like (KAMM, 1990) and (SKAKOON, 2009) talk about Exact CD and the different benefits obtained when using it, nevertheless they also express the necessity of overconstraint since sometimes big structures require certain strength and stability that can only be achieved using Redundant CD. Self-aligning appears in the literature as a way to allow extra freedom to the joints of a mechanism in order to cancel the effects of an overconstraint which affects the reliability of the mechanism.

An example of self-aligning is shown in Figure 2, the wheel illustrated can be normally found in a supermarket trolley, it helps the trolley turn in any direction the driver wants to go. This is because the design of such a wheel allows two rotations, one in the direction the driver wants to go and another rotation by the complete wheel allowing such alignment of the force and the direction of the wheel. Self-aligning features are to be proposed to the design, in order to achieve strength and stability and to eliminate internal stresses and deformations brought by the overconstraint state.

1.4 CHAPTER OVERVIEW

In this introductory chapter, the notion of kinematic analysis and CD have been discussed, since they offer a basis to understand what will be developed in the following chapters. In chapter 2 a review of the existing literature defines what a mechanism is, what its components are and a deeper exploration of the CD concept is given. In the same chapter Graph Theory is presented as a way to represent the topology of a mechanism, and the Screw Theory is explained and then used as a tool for kinematic and static analysis. In chapter 3 the Davies Method (DAVIES, 2006), the Path Method (SHUKLA; WHITNEY, 2005) and the methodology described by Reshetov in (RESHETOV, 1979) are described and compared. In chapter 4, the Davies Method is used to form a network matrix that will help in the analysis made by the proposed algorithms; other algorithms used are explained as well. In addition, the results of the analysis of one mechanism will be discussed, together with some of the self aligning proposals. Chapter 5 concludes the dissertation, summarizing important concepts of this research and its results and pointing out areas of opportunity for future work.

2 LITERATURE REVIEW

In this chapter a review of the literature related to machine and mechanism science is presented according to, and based on: (POLLARD, 1933), (DAVIES, 1968), (DAVIES, 1970), (HUNT, 1978), (RESHETOV, 1979), (KAMM, 1990), (DAVIES, 1995a), (DAVIES, 1995b), (BLANDING, 1999), (DAVIES, 2000), (SKAKOON, 2000), (WHITNEY, 2004) and (GOGU, 2008). In order to evaluate the theory related to mechanism, this chapter is divided into three sections. In section 2.1, the text explains what a mechanism, its spatiality and its characteristics, specifically the freedoms and constraints allowed by the joints and the mobility of the kinematic chain. Section 2.2 is an introduction to Graph Theory and Screw Theory, they are reviewed so they can be used as tools for kinematic and static analysis, using the methodology explained in the next chapter. In section 2.3, Constraint Design (CD) describes the different states of constraint in mechanisms. In the following chapters, these concepts will be used to comprehend the results of the examples in chapter 4.

2.1 MECHANISM AND KINEMATIC CHAIN

Bodies in mechanisms are known as *pairing elements* and they can be formed with different materials, shapes and sizes; however for designing purposes they are classified into either rigid or flexible bodies. *Rigid bodies* are considered to be strong enough to maintain their dimensions when force is applied, any infinitesimal deformation in the material is unconsidered for designing purposes. In general, rigid bodies maintain a constant distance between two points at all time. *Flexible bodies*, on the other hand, like springs, belts or bands, deform when force is applied. The term flexible element is used to define a belt (IONESCU, 2003). An important characteristic of the rigid bodies is their capacity to pair with other rigid bodies, therefore the term *pairing elements*. A pairing element is considered to be a *link* in the kinematic chain, capable of coupling with other links rigid enough to transmit motion to other pairing elements. The *joint* couples two bodies by restricting their ability to move, they can be recognized as the articulations that allow motion between the elements (SKAKOON, 2009). In this text, rigid bodies will be used, unless otherwise specified.

In (HUNT, 1978) the interpretation of *mechanism* is *"means of*

transmitting, controlling, or constraining relative movement". The author mentions that, despite the fact mechanisms operating electrically, magnetically, pneumatically and hydraulically are included in the definition above, *mechanisms* will be defined as *rigid bodies* that are connected by joints that allow relative motion in desired directions. In (TSAI, 2001), the *kinematic chain* is presented as an *"assembly of links or rigid bodies, that are connected by joints"*, and a mechanism is *"when one of the links in a kinematic chain is fixed to the ground or base"*. Figure 3 presents different mechanisms in each row, the left column illustrates a schematic representation, the right column illustrates the kinematic chain structure. According to Tsai's definition, left and right columns are both considered mechanisms since both have a fixed defined base. In the same figure, rigid bodies are identified with numbers and the joints, with letters.

A link in the kinematic chain denotes the number of connections it has with other links. A binary link is a pairing element capable of pairing with two other pairing elements, a ternary with three others, a quaternary with four and so on. A series of connected links is considered to be an *open-loop kinematic chain*. If the last link in the series connects with the first link then it will be a *closed-loop kinematic chain*, simply called a *loop*.

Following figure 3, the mechanism in b) has one loop in its kinematic chain representation, while the mechanism in c) possesses three loops in its kinematic chain representation. Structurally, mechanisms with closed loop kinematic chains present more stability and force when executing tasks than open loop kinematic chains (SICILIANO et al., 2009). This research will analyze closed loop kinematic chains in order to find the freedom and constraint relationship allowed by the joints on the mechanism.

In the following subsections the research explains a spatial concept: the dimension of the space where the motion of the mechanism occurs as well as the space where constraints and freedoms act in paired elements on the degrees of freedom (*dof*) of the joints.

2.1.1 Freedom and Constraint

In a three dimensional space, the origin is located at the point $\mathcal{O}_{xyz}$, here the bodies are allowed to translate and rotate freely along any axis. A body without constraints has six *degrees of freedom* (*dof*), (POLLARD, 1933) this is considered to be a kinematic theorem. Trans-

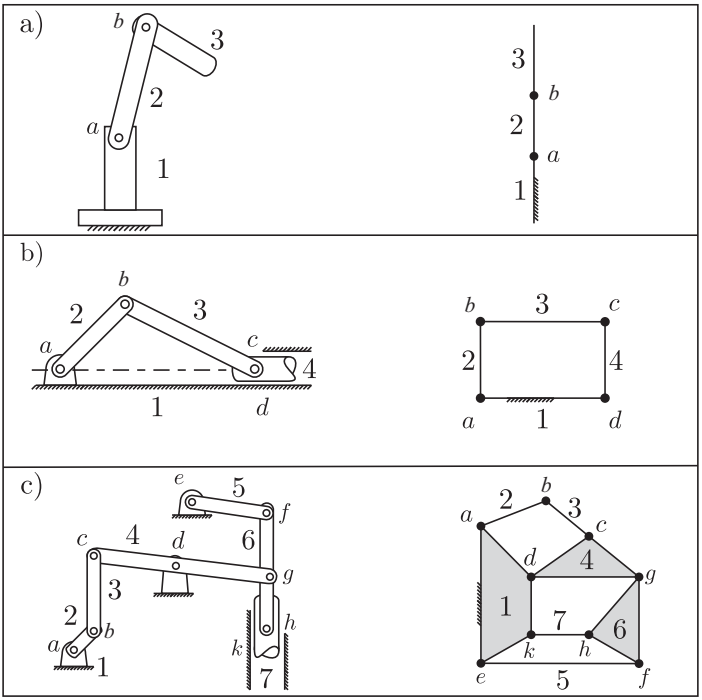


Figure 3 – Illustration of mechanisms and their kinematic chain representation, a) (TSAI, 1999) Fig. 1.2, b) (TSAI, 2001) Fig. 1.6 and c) (BROWN, 1995) Fig. 169.

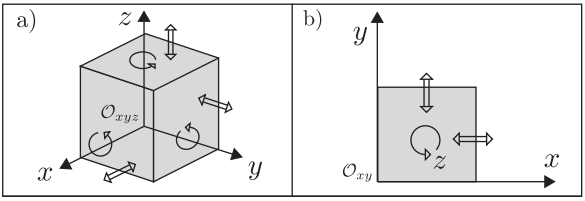


Figure 4 – a) The three dimensional coordinate system with origin at $\mathcal{O}_{xyz}$ and the six freedoms allowed, in b) the two dimensional coordinate system with origin at $\mathcal{O}_{xy}$ and the three freedoms.

lations and rotations are considered to be *freedoms* (f) of the unconstrained body. To restrict motion, (c) *constraints* are used, one for each of the six *dof* that the body has. Each degree of freedom should be constrained by an independent restriction such that, the body has six *degrees of constraint* (*doc*) to impede all motion.

The parameter λ refers to the order of the system. Six motion components are represented by "*six homogeneous coordinates, each one corresponding to an independent degree of freedom*" (HUNT, 1978). The total amount of freedoms and constraints sum the order of the system as shown in equation 2.1. The value can be any between zero and six, for $\lambda = 0$ the body is fully constrained, it has no motion since, again, all six *dof* were perfectly constrained by six *doc*.

$$\lambda = f + c \qquad (0 \leq \lambda \leq 6) \tag{2.1}$$

Figure 4 illustrates a three dimensional space in a). The figure identifies the origin point $\mathcal{O}_{xyz}$, and the six freedoms. The three round arrows indicate the rotation and the three double-line arrows indicate the translations along each axis. The sum of these freedoms gives $\lambda = 6$, therefore the represented element is unconstrained.

In b), the figure shows a two dimensional space with origin at the point $\mathcal{O}_{xy}$. Three freedoms allow a body one rotation, indicated by the round arrow around the z axis, and two possible translations: one in the x direction and other in the y direction. The system order for such coordinates will be $\lambda = 3$

Mechanisms can be designed to work in different dimensions, *spatial mechanisms* use the six degrees of freedom and they are $\lambda = 6$, they allow six possible *dof*, while *planar mechanisms* have $\lambda = 3$, they move in a two axes coordinate system, as shown in Figure 4. The system

order for gears is $\lambda = 2$, one rotation and one translation along the axis of the shaft are the only freedoms allowed by such mechanisms. These are the more common values for λ, nevertheless the mechanical systems can be any between one and six, otherwise if $\lambda = 0$ the mechanism will be a rigid structure without motion.

Table 1 presents a copy of table 2.1 found in (GOGU, 2008). The author includes in it different terms used in the literature when referring to λ in the Grübler and Kutzbach equation for determining the mobility of the mechanism.

Terminology	References
Motion parameter	Somov 1887
Mechanism category	Hochman 1890
Rank of linear set of screws	Voinea and Atanasiu 1959
Link mobility	Voinea and Atanasiu 1960
Relative infinitesimal displacement of two bodies	Hunt 1967
Connectivity of the complex joint between two bodies	Waldron 1966
Connectivity of the instantaneous screw system of two bodies	Davies and Primrose 1971
Degree of freedom of the complex joint between two bodies	Hervé 1978
Relative freedom between links	Baker 1980
Complex connectivity	Baker 1980
Internal freedom	Baker 1981
Connectivity between a pair of members	Phillips 1984
Freedom of the complex joint between a pair of members	Phillips 1984
Rank of the kinematic space of the members	Dudită and Diaconescu 1987
Kinematic constraints between two bodies	Fanghella 1988
Connectivity	Fanghella and Galletti 1994
Dimension of the space of twists between two bodies	Fayer 1995a
Link connectivity	Shoham and Roth 1997
Degree of freedom of the mechanism with an output member	Zhao et al. 2004
Spatiality	Gogu 2005b-e

Table 1 – Gogu's terminology for the parameter λ, (GOGU, 2008).

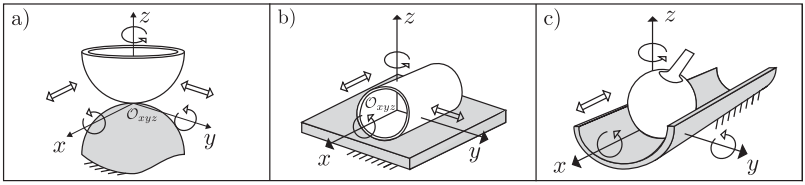


Figure 5 – Higher pairs: a) point contact, b) line contact and c) curve contact, (HUNT, 1978).

2.1.2 Joints: Kinematic Pairs

The connection between two bodies i and j will impose c constraints in the pairing elements to obtain net *dof* f_{ij} and constraint c_{ij} between the two bodies. According to (HUNT, 1978), a *kinematic pair*, or just *pair*, means that only two bodies are being jointed and they guarantee continuous contact between the bodies during relative motion. In (IONESCU, 2003), a *kinematic connection* is defined as a *"mechanical model of the connection of two pairing elements having relative motion of a certain type and degree of freedom"*. The latter is mentioned because there is a difference in the joints of a mechanism, meaning that either they can be just the two elements guaranteeing contact between each other or a mechanical model is responsible for the connection of the two pairing elements. One example is the *universal joint*, where two rotative *dof* in perpendicular axis are allowed. This dissertation text will only deal with the kinematic pairs and their characteristics, since kinematic connections are created using kinematic pairs.

The *dof* and *doc* of a joint are considered important parameters, since they indicate how many degrees of freedom f are allowed and how many constraints c are being imposed on the pairing elements. These parameters of the kinematic pairs are defined by the type of contact between the elements and the number of constraints, as seen above. Depending on the type of contact, kinematic pairs can be divided into *higher pairs* and *lower pairs* (HUNT, 1978). Higher pairs are those where the contact between the elements is defined by a point, a line or a curve (see Figure 5). In the literature and in this research, a practical consideration is made about higher pairs: since pure point contact does not exist, there is always a certain amount of material that squashes to form a tiny surface that will be in contact, therefore for designing

purposes, the contact between two elements takes place neglecting material deformation. For this reason, mechanisms with higher pairs like gears or cams have a high concentration of force between the bodies in contact. To avoid such concentration, the force between the bodies can be spread in a surface, as is the case with lower pairs.

Lower pair means that there is a surface contact between the elements. There are six lower kinematic pairs (HUNT, 1978): spherical pair (S-pair), planar pair (E-pair), cylindrical pair (C-pair), turning pair (R-pair), prismatic pair (P-pair) and screw pair (H-pair), all identified in Figure 6, with their symbolic representations.

The spherical pair (S-pair), identified in Figure by a), allows three rotations, one on each axis. The planar pair (E-pair), in b), also allows three freedoms, two translations and one rotation on the axis perpendicular to the surfaces in contact. In the cylindrical pair (C-pair), shown in c), four constraints are imposed from the fixed element to the moving element, allowing it to rotate and translate on the same axis independently. The rotative pair (R-pair), in d), allows one rotation by the constraint of the other five *dof* of the pairing element. The prismatic pair (P-pair), in e), permits one translation depending on where the axis is located. The screw pair (H-pair), in f) admits two degrees of freedom as the combination of a translation and a rotation acting at the same time in the same axis.

Table 2 summarizes both types of kinematic pair and indicates the number of *doc* and *dof* that each type of pair has in terms of rotations and translations. Each rotation is represented by the letter (R) and each translation by the letter (T). Over time, these kinematic pairs in mechanisms have proven their easy alignment with the axis of the coordinate system. For the purposes of analysis, a kinematic pair can be substituted by pairs with less *dof*, so, when they are connected they can achieve the same *dof* as the substituted pair. For example the construction of an S-pair as shown in Figure 6, can be quite difficult, nevertheless, with three R-pairs, each acting on one axis, this will provide the same three rotations as the S-pair. A C-pair can be represented as the result of an R-pair and a P-pair, both acting on the same axis. The Screw Theory in section 2.2.2 explains how a translation and a rotation represent motion in space.

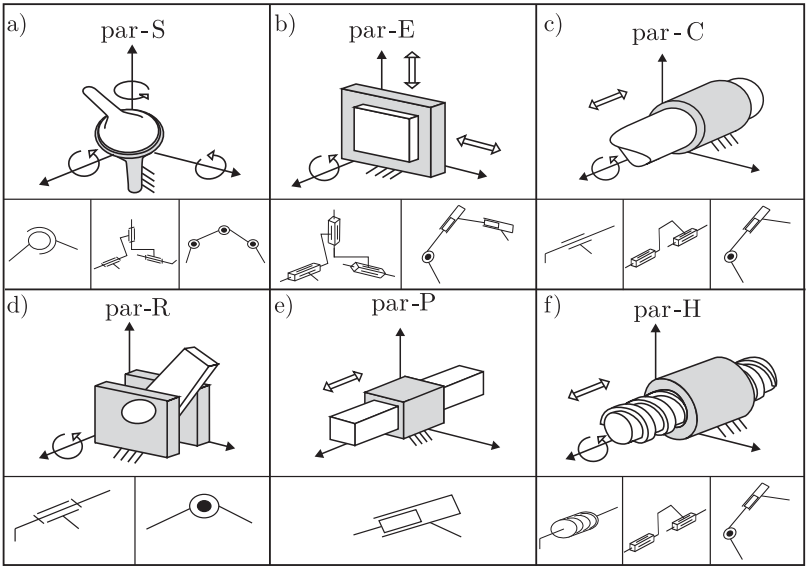


Figure 6 – Lower pairs: a) Spherical (S-pair), b) Planar pair (E-pair), c) Cylindrical pair (C-pair), d) Rotative pair (R-pair), e) Prismatic pair (P-pair) and f) Screw pair (H-pair), (HUNT, 1978).

Table 2 – Summarized *doc* and *dof* of kinematic pairs.

High pairs		Point contact	1	5	T	$RRRTT$
		Line contact	2	4	RT	$RRTT$
		Curve contact	2	4	TT	$RRRT$
Low pairs	S-pair	Spherical	3	3	TTT	RRR
	E-pair	Plane	3	3	RRT	RTT
	C-pair	Cylindric	4	2	$RRTT$	RT
	R-pair	Revolute	5	1	$RRTTT$	R
	P-pair	Prismatic	5	1	$RRRTT$	T
	H-pair	Helical	4	2	$RRTT$	RT
Type	Pair	Contact	*doc*	*dof*	Constraints	Freedoms

2.1.3 Mobility Formulae

IFToMM terminology defines the mobility of a mechanism as the number of independent coordinates required to determine the configuration of a kinematic chain or mechanism (IONESCU, 2003). Mobility M is a parameter used to know the number of independent inputs needed to drive the mechanism (GOGU, 2008). A global mobility is used as a single value and it represents the mechanism in all its positions except for its singular ones.

A local mobility represents the mechanism in an instant frame. Local mobility can be equal or less than the global mobility. The calculus of the mobility involves the type of joints and their quantity represented by j, the number or pairing elements in the kinematic chain represented by n, the spatiality λ, the constraint c, the freedoms f and the information about the position of the elements in the mechanism in order to set up equations that can be analyzed using Screw Theory (DAI et al., 2004), or linear algebra (RICO; GALLARDO; RAVANI, 2003), to perform velocity and static analysis that determine the rank of the mechanical system that will indicate the number of independent equations that determine the mobility. To avoid setting equations that take time to solve, quick formulae have been developed through time to determine the mobility (GOGU, 2008). They are based only on the number of joints (j), the number or pairing elements (n) and the order of the system λ. The equation 2.2 is taken from (HUNT, 1978) and it is based on the general mobility criteria developed by Grübler (1917) and by Kutzbach (1929). If the freedoms of the joint i are considered as independent, then the number of joints can represent the number of independent freedoms since, $\sum_{1=i}^{j} f_i$.

$$M = \lambda(n - j - 1) + j \tag{2.2}$$

2.2 REPRESENTATION OF A MECHANISM

This research uses Graph Theory and Screw Theory as tools to represent the information that characterizes a mechanism in matrices. There are different visual ways to represent mechanisms, a *schematic* or functional representation is a blueprint of the mechanism where a sketch illustrates the mechanism as close as possible to reality, as with the mechanisms shown in the left column in Figure 7 (the same closed-loop

mechanisms as in Figure 3). A structural representation also known as the kinematic chain, defined in section 2.1, represents the pairing elements with polygons where each vertex is a connection point with another pairing element. Kinematic chain representation is illustrated again, in the central column in the same Figure 7.

Graph Theory can be used to represent mechanisms, it uses nodes to represent rigid bodies, and lines or edges to connect the nodes symbolizing the kinematic pairs (DAVIES, 1995b). Section 2.2.1 explains some of the graph characteristics. In the right column of the same Figure 7, the coupling graph is presented for each mechanism. Screw Theory is then used to analyze the systems of motions and actions allowed in the mechanism. In instantaneous kinematics the freedoms allowed are considered to be linear and angular velocities and in statics the actions allowed represent the forces and the moments determining its static stability, as is explained in section 2.2.2.

2.2.1 Graph Theory Applied to Mechanisms

There are several reasons for using Graph Theory. A major one is because it simplifies the identification of the mechanism topology, since there are already implemented algorithms that can be used to analyze the structure. In (SIEK; LEE; LUMSDAINE, 2002), a complete library with graph algorithms solved and ready to use in C^{++} programming language, although this research was implemented in GNU Octave programming language. A graph represents a mechanism with (V) *vertices* that stand for the n pairing elements, and the (E) *edges* or arcs correspond to the j joint in the kinematic chain. The equation 2.3 indicates that the graph (G) is a set of vertices and nodes.

$$G = \{V, E\} \tag{2.3}$$

It is possible to represent the relation between the vertices and the nodes using the *Adjacency Matrix* and *Incidence Matrix* (CARBONI, 2008). The adjacency matrix $[A]_{(n \times n)}$ represents in its rows and columns the n links of the mechanism, therefore it is considered to be a square matrix. The adjacency matrix is formed by A_{ij} elements that indicate if the pairing element i is jointed to the pairing element j. The variable A_{ij} is a binary that represents if there is an edge between the two nodes i and j in the graph G, it can be:

$$A_{ij} = \begin{cases} 1, & \textit{if an edge exists between the nodes } i \textit{ and } j \\ 0, & \textit{if no edge exists to connect the nodes} \end{cases} \tag{2.4}$$

When the edges of a graph are orientated, the graph is called a *digraph* (CHRISTOFIDES, 1975). Orientation can be set arbitrarily, the edge can be defined with direction from node i to node j if $(i < j)$, the directed edge is called an *arc*. A digraph can be represented by the incidence matrix $[I]_{(n\times j)}$ which represents the pairing elements in its rows and the joints of the mechanism in the columns. The matrix is useful to know the topology of the kinematic chain and to detect kinematic chains that are equal (CARBONI, 2008). The element I_{ij} indicates if the pairing element i couples with another pairing element using the joint j, and the direction of the arc in the graph such that:

$$I_{ij} = \begin{cases} +1, & \textit{if the node } i \textit{ connects using edge } j \textit{ pointing to other node} \\ -1, & \textit{if the node } i \textit{ connects using edge } j \textit{ pointing into node } i \\ 0, & \textit{if no edge exists that connects the nodes} \end{cases} \tag{2.5}$$

Just like the kinematic chains, graphs can have open and closed-loop structures. The open-loop is considered to be a *path* where the edges are connected in a serial arrangement, in a *closed-path* the last edge in the series connects with the first edge. For example in a graph G with at least one closed-path or loop, it is possible to obtain a subgraph without loops, this sub-graph is called a *tree*, it includes all the vertices and as many edges as possible, the edges can be called *branches*. The edges that were removed to form the tree are called *chords*. The total of chords removed indicates the number of loops in graph G.

In Figure 7 the graphs in the right column represent the structure of the kinematic chains maintaining the same connection relationship between the pairing elements. The mechanisms represented by a graph focused on the couples is called graph G_c, it represents in its vertices the pairing elements, they are identified by numbers, and the edges represent the joints of the mechanism and they are identified with letters. The direction of the edges can be identified by the direction of the arrows, the chords are identified with dotted arrows, they also indicate the direction of the loop. The branches are the rest of the arrows, they form the tree of the graph.

From the coupling graph G_c a fundamental circuit matrix $[B]_{(l\times E)}$ can be obtained to make a relationship between the pairing elements

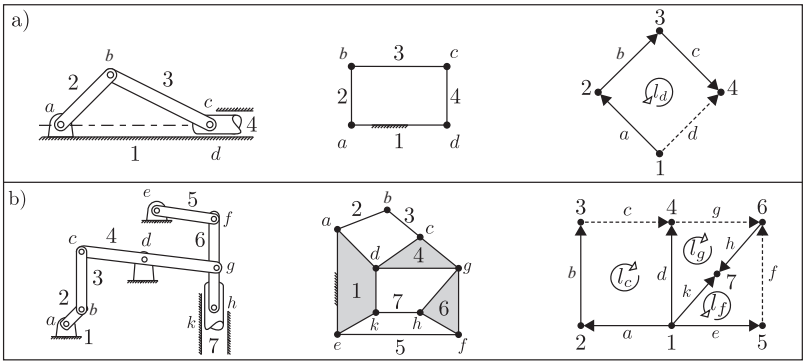


Figure 7 – Mechanisms in Figure 3 and the corresponding structural graph representation.

and the joints in each loop indicated in the element b_{ij}, if it represents the existence of E in the loop l as indicated in equation 2.6.

$$b_{ij} = \begin{cases} +1, & \textit{if the edge has the same direction as loop } i \\ -1, & \textit{if the edge does not has the same direction as the loop} \\ 0, & \textit{if no edge exists that connects the nodes} \end{cases} \tag{2.6}$$

A cutset matrix can be obtained to relate the chords, for each chord in the graph G_c. The fundamental cutset matrix $[Q]_{k\times E}$ is represented in the element q_{ij} if the branch j is included in the cutset i as indicated in 2.7. The cuts divide the graph into two subgraphs (YANG; LEE, 2007).

$$q_{ij} = \begin{cases} +1, & \textit{if the edge direction is the same as the cutset} \\ -1, & \textit{if the edge direction is opposite to the cutset} \\ 0, & \textit{is the edge does not exist in the cutset} \end{cases} \tag{2.7}$$

The circuit matrix and cutset matrix will be further detailed as required in sections 3.2.1 and 3.2.2 respectively.

2.2.2 Screw Theory

Screw Theory as presented in this text is based on (MARTINS, 2002), (DAVIES, 1995a) and (CAZANGI, 2008), since they use screws

to represent mechanisms. Screw Theory stipulates that a rigid body can be represented by a translation and/or a rotation about an axis (BALL, 1998). The screws act in an axis found by Mozzy (1763) for instantaneous kinematics and in an axis found by Poinsot (1806) for statics. In instantaneous kinematics translation motion is considered to be a linear velocity vector ($\vec{v}$) and the rotation is considered to be an angular velocity vector ($\vec{\omega}$). In statics the translation is considered to be a force vector ($\vec{R}_R$) and a rotation is represented by a moment vector ($\vec{T}_R$) as presented in (CAZANGI, 2008).

The screw is represented by (\$), and is created from a geometrical point p and a fixed straight geometrical line or axis with origin at the point $\mathcal{O}_{xyz}$. The use of vectors help relate the geometrical entities, this simplifies the visualization and defines each coordinate axis in a matrix. Two vectors are defined to relate the point, the axis and the origin. A *free vector* $\vec{v}$, which needs direction, sense, and length to be defined, and a *line vector* $\vec{u}$, which is defined in its direction, same as the axis. The description of a screw \$ is done relating the line vector $\vec{u}$ created from point x_1 to point x_2 defining the axis of the screw. A radius vector $\vec{r}$, goes from the axis to the point or particle p. In physics a vector $\vec{v}$ represents the moment of a particle and is obtained from the cross product of the vector $\vec{r}$, that describes the distance from the point p to the vector $\vec{u}$, the screw \$ is formed by both vectors as in equation 2.8, where $\vec{v} = \vec{r} \times \vec{u}$.

$$\$ = \begin{bmatrix} \vec{u} \\ \vec{v} \end{bmatrix} \tag{2.8}$$

The screw \$ can also be obtained from vector $\vec{S}$ (line vector) on the direction of the screw axis and the position vector $\vec{S_0}$ which is a vector from the origin $\mathcal{O}_{xyz}$ to the closest point on the axis of the screw. Figure 26 illustrates the geometrical representation of the motion screw, and Figure 25 represents the action screw. The pitch (h) is a constant obtained from the relation between translational and rotational motions, both allowed by the screw. In this way the screw is represented by equation 2.9 as:

$$\$ = \begin{bmatrix} \vec{S} \\ \vec{S_0} \times \vec{S} + h\vec{S} \end{bmatrix} \tag{2.9}$$

In kinematics the screw is called a *twist*. The twist is represented by $\$^M$, were M stands for motion. In statics the screw is called a *wrench* and it is represented by $\$^A$ where A stands for action.

Plücker coordinates are used in Screw Theory to represent the coordinates of the screws (DAVIES, 2000). In the following sections kinematic and static screws will be explored in greater depth.

A unitary screw $\hat{\$}$ can be obtained by separating the directory cosines, which indicate the angle between the vector and the coordinate system axes, from a constant that magnifies the size of the screw. We will define this magnitude for kinematics as ϕ and for statics as ψ. The screw is formed by $\$^M = \hat{\$}^M \phi$ and $\$^A = \hat{\$}^A \psi$ as is explained in Appendices A and B.

2.3 CONSTRAINT DESIGN (CD)

As mentioned in the introduction of this text, CD is an important concept to consider when designing a mechanism, as it allows the engineer to control motion by knowing and determining each constraint individually. The result is a reliable and fully determined mechanism.

As seen before, there are six constraints that will determine the state of a fixture or kinematic connection. A design is said to be an *Exact* CD when the connection between two elements has all six freedom individually constrained. In (BLANDING, 1999) and in (KAMM, 1990) it is mentioned that the understanding of motion leads to high performance mechanisms at low cost. In an article by (SZYDLOWSKI, 2000) where including CD as a subject in the engineering curricula at universities is proposed, the author mentions that: *"Practical and inexpensive experiments done by the students in the classroom help build intuition by helping the students understand the concept of mobility, redundant constraints and self-alignment".* In CD, mechanisms and their kinematic chains are analyzed, based on the freedoms and constraints that each joint has. The type of constraints imposed on the joints determines not only the mobility of the mechanism, but also determines its assembly constraints (WHITNEY et al., 1999).

If Exact CD is applied using the minimum amount of constraints possible then it is said to be a *Min* CD. An overconstraint mechanism refers to redundant constraints when preventing a certain *dof*, therefore this is called a *Redundant* CD or *Red* CD. An *underconstrained* design would then be when joints in the mechanism have *dof* that are not constrained, causing them to undesirably move freely and without control. *Self-aligning mechanisms* are those kinematic chains that allow extra *dof*, which eliminates the redundancy of an overconstraint mechanism.

2.3.1 Minimum CD / Exact CD

Minimum CD or MinCD is presented by (KAMM, 1990) as a way to "*provide only the minimum number of constraints needed to permit the freedoms desired and no other freedoms*" (KAMM, 1990). It is stated in (SKAKOON, 2009) that "Exact constraint means constraining the six degrees of freedom, no more and no less, to obtain the desired structure, or leaving one or more unconstrained to obtain the desired motion". In Min CD and Exact CD the objective is then basically the same, to constrain motion with independent constraints, one for each component of motion. Precision engineering may be related to the design of mechanisms using Exact CD (SCHELLEKENS et al., 1998). Douglas Blanding in "Exact Constraint: Machine Design Using Kinematic Principles" published a set of rules for Exact CD, althought there is no formal methodology for such type of design. The rules are presented in table 3 as summarized in (HAMMOND, 2004).

Rules for $\lambda = 3$	No two constraints are collinear No four constraints are in a single plane No tree constraints are parallel No tree constraints intersect at a point
Rules for $\lambda = 6$	No four constraints are parallel No four constraints intersect at a point No four constraints are in the same plane

Table 3 – Rules for Exact CD, taken from (HAMMOND, 2004), p.20

Figure 8 illustrates the Kelvin's coupling, this example is widely used in the literature because it illustrates how with three points of contact the coupling gets perfectly constrained. In the figure the pairing element (1) has in a) a trihedral hollow that constrains motion with tree orthogonal surfaces, in b) a right angled groove with two plane surfaces, and the sixth plane is the horizontal surface of contact at point c). The pairing element (2) has three spherical surfaces to make contact with the other pairing element. This form of coupling makes it very easy to notice when the coupling is not properly assembled.

2.3.2 Redundant CD / Over-constraint

Internal stress and deformation are mentioned in the literature as the result of constraining an element in more than six ways (DAV-

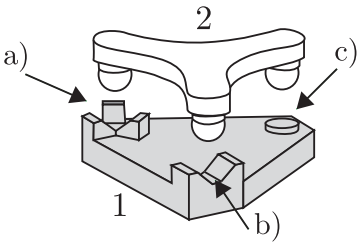


Figure 8 – Exact Constraint Coupling (Kelvin's Coupling), (HALE; SLOCUM, 2001).

IES, 1970). A redundant CD mechanism is in a state of overconstraint where the *dof* are prevented more than once. This means, from the six components of motion in ($\lambda = 6$), one or more have been constrained more than once. The use of force for joining elements that are overconstrained causes deformation of the elements during assembly. "*If the constraints between two members exceed six, the members will suffer unnecessary elastic deformation*" (POLLARD, 1933). To solve the mathematical problem, deformation equations are used to completely determine the static equilibrium, where the number of these equations indicate the number of redundant constraints (RESHETOV, 1979). An example of an overconstrained assembly is illustrated in Figure 12, where the shape of the tab does not allow the screw to secure it to the fixed frame. If the screw is forced, then the two shown forces will act and a moment will cause a deformation on the tab. In section 2.3.4 a possible solution to this problem will be presented.

Deformation is considered to be desired or undesired, Redundant CD considers deformation as part of the design project. The methodology considers controlling the deformation by projecting it on an intended direction. Redundant CD is used to design mechanisms or fixtures that require more stability (SKAKOON, 2009). For example the five wheel chair illustrated in a) in Figure 9 gives more stability to the user than a three wheel chair. Four legged tables, like the one shown in b) in the same figure, are normally used over three legged tables since they can deform along the diagonal of the table. Allowing the four legs to be in contact with the floor brings stability to the weigh that is being supported (KAMM, 1990).

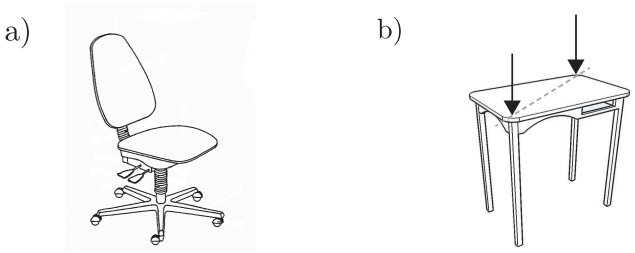


Figure 9 – Examples of Red CD, they allow more stability to the design, (KAMM, 1990).

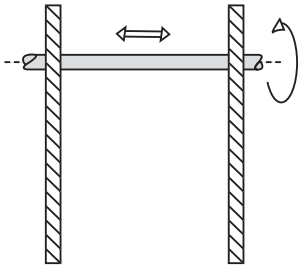


Figure 10 – Unconstrained shaft, supported by two walls.

2.3.3 Under-constraint

As mentioned in the beginning of this section, to leave a mechanism with undetermined motions can be dangerous (WHITNEY et al., 1999). An undetermined *dof* in a pairing element causes the pairing element to move freely, independently of the rest of the mechanism.

This means motion can occur in directions where it should not move and freedoms are left without enough constraints (SMITH, 2001). An example that explains under-constraint is illustrated in Figure 10. The shaft is relatively fixed by the two fixed walls, the shaft is no longer capable of going up and down and inside and outside the paper, nevertheless it is still capable of translating and rotating along its axis, both indicated by the direction of the arrows. The rotation of the shaft is evident to the designer, but if the shaft is used to support a load, then the translational freedom can be considered as a dangerous mobility.

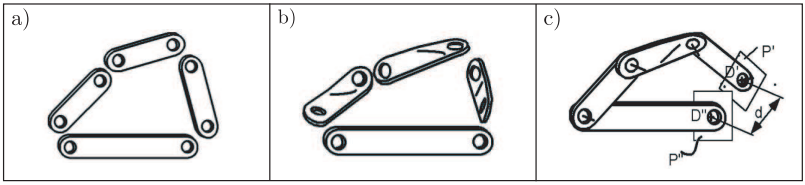


Figure 11 – Four bar mechanism with serious geometric errors that impede the assembly, (SZYDLOWSKI, 2000).

2.3.4 Self-aligning

In order to eliminate redundant constraints that affect the accuracy of a mechanism self-aligning may be defined as "*freedoms that permit (a) mechanism to align parts and axes under the working torques* (KAMM, 1990). Gimbals, universal joints, self-aligning bearings, spherical joints among self-aligning mechanical elements are used to adjust any existent difference between the axis of the load and the axis of motion. In (SZYDLOWSKI, 2000) the emphasis for self-aligning when designing a mechanism is on making the assembly possible. It is mentioned that possible variations between a designed and a machined parts can cause pre loads in the assembly or deformation of the parts when force is required to assemble the parts. Figure 11 illustrates a four bar mechanism. From left to right it shows the way the parts were designed, in the center is shown how a really poor manufacturing process made the parts, in the end the assembly process reports a distance (d) missing for the mechanism to close without the use of force. This example illustrates the necessity of self-aligning. (RESHETOV, 1979) explains that a way to achieve assembly in overconstrained mechanisms like the one in the figure, is by swapping the kinematic pairs used with others with a higher *dof*. Additionally, the author mentions that kinematic connections between the parts may also be used.

Self-aligning has been researched by (DOWNEY; PARKINSON; CHASE, 2003), where the term "smart assembly" is defined as: "*features, not otherwise required by the function of the design, which allow the design to absorb or cancel out the effects of variation.*" This is a major advantage because "the value and location of forces are independent of the tolerances, clearances and preloads" (RESHETOV, 1979). The mechanisms that present self-aligning characteristics eliminate overconstraint and its consequences. The following two examples illustrate the idea of

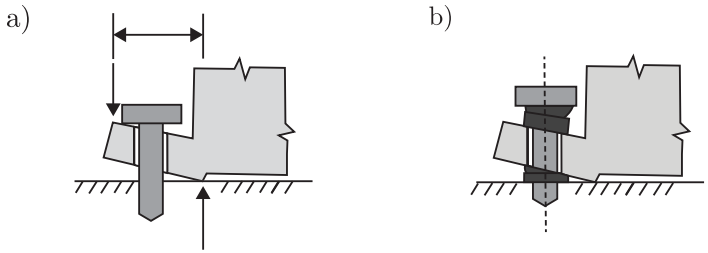


Figure 12 – Exagerated image of a) Overconstrainted bolted foot and b) self-aligning bolted foot with spherical washers, (KAMM, 1990).

what is self-aligning.

Self-aligning can be considered to act in a passive or inactive way. Passive self-aligning is used to ease the assembly process by eliminating overconstraint (DOWNEY; PARKINSON; CHASE, 2003). An example is shown in Figure 12: it presents a solution for the overconstrained bolted foot. By using spherical washers it is possible to align all the constraints such that non use of force is required to pair the foot with the fixed frame.

Active self-aligning mechanisms "*can adapt to variation during the life of the design*" (DOWNEY; PARKINSON; CHASE, 2003). Figure 13 illustrates an example of an active self-aligning connection. This type of connection can be found in garage doors: the roller is trapped inside the guide making the door go up and down, the cylindrical sleeve keeps the door aligned even if the fixed guides are misaligned, the sleeve allows a linear freedom where the cylinder of the roller can travel to cancel any variation.

In the beginning of this section it was mentioned that kinematic pairs or kinematic connections with a higher *dof* may be used in mechanisms to eliminate overconstraint. Those redundant constraints that will not affect the mobility when eliminated are called "passive" constraints. The *dof* increment between two parts should be made carefully, the mobility should not be affected by the change and no unconstrained motion components should be left in the mechanism (RESHETOV, 1979).

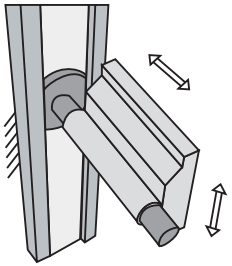


Figure 13 – active self-aligning mechanism, representation of a roller assembly from a garage door, (DOWNEY; PARKINSON; CHASE, 2003).

3 METHODOLOGY: LITERATURE REVIEW

In the research of the literature, three methodologies were found and analyzed. One of them will be used and implemented to perform the kinematic analysis developed in chapter 4. In section 3.1, the text presents a method found in (RESHETOV, 1979) and (SZYDLOWSKI, 2000), which is here called the table method, where the authors determine the state of constraint of mechanisms, based on the quantity and direction of the freedoms allowed by each joint. Section 3.2 presents the Davies method, based on (DAVIES, 2006) and (DAVIES, 2000), where the author explains a methodology using Screw Theory and Graph Theory for an instantaneous kinematic and static analysis of the mechanisms. In section 3.3 the path method is described as presented in (SHUKLA; WHITNEY, 2005). This methodology considers all the possible paths from one pairing element to another to describe the state of constraint of the mechanism. Section 3.4 concludes the chapter mentioning the differences between the methods and specifying which one will be used in the next chapter.

3.1 RESHETOV: TABLE METHOD

This section describes a methodology to analyze a mechanism and determine its mobility and overconstraints. The objective of such analysis is to eliminate overconstraints by augmenting the *dof* of the joints in order to improve the design and make it more reliable (RESHETOV, 1979). The method counts the quantity of degrees of freedom allowed by each independent joint, and relates them to the loop they belong to.

The equation 3.1 was proposed by (SZYDLOWSKI, 2000) to determine the quantity of (l) loops based on the n pairing elements and the j joints in the mechanism. The method considers each joint to be represented by a rotation $\vec{R}$ vector and a translation $\vec{T}$ vector. Equations in 3.2 represent the sum of all rotation and linear velocities in the mechanism in a $\mathcal{O}_{xyz}$ axes system.

$$l = j - (n - 1) \tag{3.1}$$

$$\begin{aligned}
\vec{R}_x &= \sum_{i=1}^{n} \vec{R}_{xi} \quad \vec{T}_x = \sum_{i=1}^{n} \vec{T}_{xi} \\
\vec{R}_y &= \sum_{i=1}^{n} \vec{R}_{yi} \quad \vec{T}_y = \sum_{i=1}^{n} \vec{T}_{yi} \\
\vec{R}_z &= \sum_{i=1}^{n} \vec{R}_{zi} \quad \vec{T}_z = \sum_{i=1}^{n} \vec{T}_{zi}
\end{aligned} \tag{3.2}$$

The method considers that "*a loop can close without redundant constraints if all six freedoms are present*" (RESHETOV, 1979). If a linear freedom is missing in any axis, a rotative freedom can replace the missing linear freedom, to eliminate overconstraint. "*For a single-loop mechanism, the presence of all the three angular mobilities is a necessary condition for the loop to close without prestrains*" (RESHETOV, 1979). These two conditions must be met for the mechanism to be kinematically designed. Equations 3.3 and 3.4 summarize such conditions.

$$\begin{aligned}
\vec{R}_x &= 1 \quad \vec{T}_x = 1 \\
\vec{R}_y &= 1 \quad \vec{T}_y = 1 \\
\vec{R}_z &= 1 \quad \vec{T}_z = 1
\end{aligned} \tag{3.3}$$

$$\vec{R}_x \geq 1 \quad \vec{R}_y \geq 1 \quad \vec{R}_z \geq 1 \tag{3.4}$$

Linear freedom can be obtained by rotating the links in a perpendicular axis (RESHETOV, 1979). To replace a linear freedom, it has to be in a perpendicular axis to the axis of the rotative freedom that can be used, this is how condition 3.4 is used to satisfy the first condition. Once the replacements of the missing translations is done, the number of freedoms still missing indicates the overconstraints. The number of extra freedoms, after satisfying the first condition, indicates the extent of the mobility of the mechanism.

Figure 14 illustrates this method by analyzing two different mechanisms. The figure shows, from left to right, first the mechanism, then a graph representation, a list specifying the type of joints used, and the solution table of the analysis. In row *a*) the slider-crank mechanism has three R-pairs acting in the z axis and one C-pair acting in the x axis. The table at the right, shows the total of R rotations and T translations in each axis. Filling this table is the first step of the method. The

second step is analyzing the results, noticing that in this case, there are no rotations in the y axis and no translations in the y and z axis, therefore not satisfying the conditions 3.3 and 3.4. To satisfy such conditions, and as a third step, the extra rotations can be used to replace the lack of translations. In this case an extra rotation, the one in joint b for example can be used to replace the translation in the y axis, but it can not do it for the z, since it would not replace a translation in a perpendicular axis. This step is represented in the figure with the double line arrow. Once all the substitutions were done, the remaining extra freedoms will be considered as loop mobility, and those that remained zero, will be considered to be the number of constraints in the mechanism. The result of the method shows that for the slider-crank mechanism presented with such types of joints, an extra freedom that was not used to loop-closure indicates the mobility from the rotations in the z axis as indicated by the arrow pointing up, and two constraints from a rotation in y and a translation in z axis, as shown by the arrows pointing down. In this way it is possible to identify the freedoms that are being overconstrained.

An option to solve the overconstraint state is presented in row b) of the same figure 14. It is the same slider-crank mechanism, the difference is that there is an S-pair acting in the joint identified as b, such pair allows rotation in the three axes. The kinematic chain in the middle remains the same, since the number of pairing elements and joints did not change. In the table at the right of the row, the sum of rotations and translations reflects two rotations in the x axis, one in the y axis and again three in the z axis. The slide in joint d allows the translation seen in the x axis, as it did before in row a). Observe that translations in the y and z axes are still absent in the count, and extra freedoms from the rotations in x and z will replace such missing translations, since condition 3.4 is satisfied. The translation in the y axis will be replaced by the rotation in the z axis from joint d and the rotation in the x axis allowed by joint b will make up for the translation missing in the z axis. Both replacements are shown by the double lined arrows and they satisfy the perpendicularity condition for the missing translations. In this way, condition 3.3 is satisfied and the closure of the loop will be easy without redundant constraints. The arrow pointing up from the rotation in the z axis indicates that there is one extra freedom, apart from those required for loop closure, to be counted in the mobility of the loop.

For mechanisms with more than one loop the method repeats the steps mentioned above for each loop in the mechanism. A joint should

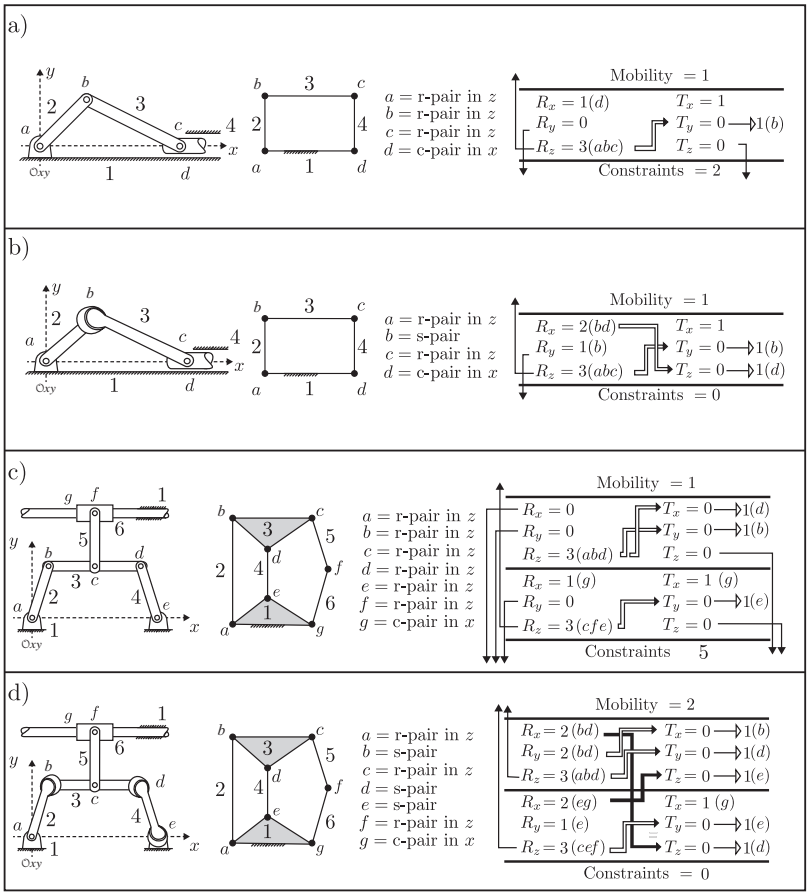


Figure 14 – Table Method applied to a four bar mechanism with: *a*) four rotative joints, *b*) one spherical and three rotative joints, and *c*) two spherical and two rotative joints.

not be counted twice, since those joints that are shared by two loops are able to replace missing linear freedoms in both loops (SZYDLOWSKI, 2000). As an example of this, again in Figure 14, rows c) and d) illustrate an example of how the replacement between loops is done. The mechanism in the third row *c*) has two loops, each indicated in each of the two divisions of the solution table. The upper part of the table is formed by the joints *abde*, which are all R-pair joints acting in the z axis. The bottom part of the table indicates a loop formed by the joints $cdefg$ which are R-pairs except for joint g which is a C-pair in x. The joints d and e are shared by both loops, joint d will be considered to be part of the first loop and joint e part of the second loop. First, in row c) the solution table represents the sum of the freedoms of the joints. The upper part of the table counts three rotative freedoms in z allowed by the joints *abd*. At the bottom, the table counts four rotative freedoms, one in x and three in z, one translation is also allowed by joint g in x axis. At first, eight freedoms are missing in the solution table from both loops. Extra rotations in z are used to replace missing translation in the perpendicular axis, decreasing the number of constraints to five, as the arrows pointing down indicate. A remaining extra rotation, with an arrow pointing up, indicates the mobility of the mechanism. To eliminate the overconstraints, the method suggests an increase in the *dof* of the joints.

Row d) presents a configuration of joints capable of eliminating the overconstraints. Before increasing the *dof* of the joints, (RESHETOV, 1979) mentions a rule for avoiding underconstraintment, this is "*a mobility s to be allocated to a loop where it eliminates a redundant constraint ad not to one where it yields local mobility.*" Joints *b*, *d* and *e* have been changed to S-pairs, increasing the *dof*. The solution table manages to replace the missing translations satisfying condition 3.3, leaving the mechanism with zero overconstraints and two mobilities. This allows the mechanism to be assembled without concern about how poor the manufacture process of the pairing elements is.

As described above,Figure 14 is used to illustrate the method proposed by (RESHETOV, 1979) and again found in (SZYDLOWSKI, 2000) where the objective is to propose a self-aligning mechanism by fulfilling conditions 3.3 and 3.4. The method presented in this section was not computationally developed by this research, but it is presented here to illustrate how self-aligning can be achieved if certain kinematic pairs are replaced by other kinematic pairs that allow more *dof*.

3.2 DAVIES METHOD

The Davies method presented in this text is based on the different articles of T. H. Davies. The author makes an analogy between electrical and mechanical systems, as mentioned in section 2.2, he explains that *"In coupling networks action screws and motion screws are analogous to electrical current and potential difference respectively in electrical network"* (DAVIES, 1995a). For mechanical systems the algebraic sum of twists in any loop is zero, as defined by the *circuit law*. For the wrenches the *cutset law* mentions that the sum of all of those wrenches that belong to the same cutset will be zero. The terminology used in this section is the same as in (CAZANGI, 2008), where a chronology of the Davies method is also described.

To analyze mechanisms the method uses Graph Theory to represent the structure of the mechanism and Screw Theory to create two equation systems presented in matrix form that are consistent with the system order λ of the mechanism. The method analyzes in one equation system the motions representing the linear and angular velocities of the mechanism, and in the other, the system of actions representing the forces and moments acting in the mechanism in order to determine the net degree of freedom F_N and constraint C_N of the coupling network.

3.2.1 Instantaneous Kinematics

In kinematics the focus is on the motion of the rigid bodies regardless what causes them to move, as mentioned before. In instantaneous kinematics the objective is to frame the mechanism at a certain time during its work cycle to determine its state of constraint for such instant.

The graph G_C is the coupling graph, where one edge represents one joint regardless of its *dof*. The edges representing couplings with more than one *dof* are expanded into to edges connected in series by virtual nodes, each representing one freedom f_i to create motion graph G_M. Figure 15 shows how the expansion of nodes using virtual nodes is illustrated, the graph represents an S-pair from Figure 6, such expansion is applied to the joint that has more than one *dof*, the edges of the graph represent each of the three rotations, one in each axis. The gross degree of freedom of the coupling network F is the sum of all the unitary freedoms f_i in the mechanical system, see equation 3.5.

$$F = \sum f_i \tag{3.5}$$

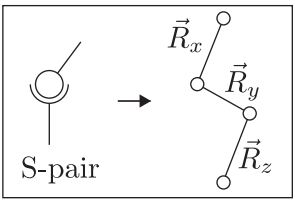


Figure 15 – Graph representation of an S-pair, using virtual nodes.

Since G_M is a graph based on G_C, the branches in G_C will continue being branches in G_M. For the expanded chords just one edge will remain being a chord and the rest of the edges will be branches, in this way the number of loops will be the same.

As seen in section 2.2.1 the coupling graph G_C has a related circuit matrix $[B]_{(l\times E)}$. The graph G_M has a related fundamental circuit matrix where all the F freedoms are included. The circuit-f matrix $[B^M]_{(l\times F)}$, is where each f_i is represented in the columns and the loops l are represented the lines of the matrix. The orientation of the graph G_M remains the same as in G_C, in this way the conditions in equation 2.6 are the same for filling the circuit-f matrix from G_M. The circuit-f matrix for the expanded graph would be as shown in equation 3.6.

$$[B^M]_{(l\times F)} = \begin{bmatrix} B^M_{i,1} & B^M_{i,2} & \cdots & B^M_{i,F} \end{bmatrix}_{(1,F)} \tag{3.6}$$

The motion matrix $[M_D]$ of the kinematic chain is represented by screw motions $\$^M$, specifically, a unitary twist $\hat{\$}^M$ is used to represent a unitary freedom f_i and gather them all in a unit matrix of direct couplings $[\hat{M}_D]_{(\lambda,F)}$. The matrix represents the twists in the columns, see equation 3.7, and each $\hat{\$}^M$ has the same number of lines as the order of the screw system, since it is the system order.

$$[\hat{M}_D]_{(\lambda,F)} = \begin{bmatrix} \hat{\$}^M_a & \hat{\$}^M_b & \cdots & \hat{\$}^M_F \end{bmatrix} \tag{3.7}$$

The magnitude vector that contains the greatness of each twist is represented in equation 3.8, where $\{\vec{\Phi}\}_{(F\times 1)}$ joins the magnitude of the motion system.

$$\{\vec{\Phi}\}_{(F\times 1)} = \begin{Bmatrix} \phi_a \\ \phi_b \\ \vdots \\ \phi_F \end{Bmatrix} \tag{3.8}$$

The motions of a mechanism are related by the unit network motion matrix $[\hat{M}_N]_{(\lambda.l\times F)}$, which combines the circuits and the twists. To form the matrix, diagonalizing each line of the circuit-f matrix $[B]_{(l\times E)}$ is required, such that a squared sub-matrix $[B_i^M]_{(F\times F)}$ is obtained as shown in equation 3.9

$$[B_i^M]_{(F\times F)} = \text{diag}\left\{ \left[B^M\right]_{(i,F)} \right\} \tag{3.9}$$

The method multiplies the unit matrix of direct couplings $[\hat{M}_D]_{(\lambda,F)}$ with every diagonalized matrix of the circuit-f matrix $[B_i^M]_{(F\times F)}$. Each resultant matrix relates the motions of the direct coupling in each loop with the motions of the other loops. This is to maintain the order of the loops as in $[B^M]_{(l\times F)}$ in order to arrange each resultant matrix under the other resultant matrix to match the loop sequence arrangement as in the circuit-f matrix. The equation 3.10 shows how the network unit motion matrix is created from the unit matrix of direct couplings and the circuit-f matrix:

$$[\hat{M}_N]_{(\lambda.l\times F)} = \begin{bmatrix} \left[\hat{M}_D\right]_{(\lambda\times F)} * \left[B_1^M\right]_{(F\times F)} \\ \left[\hat{M}_D\right]_{(\lambda\times F)} * \left[B_2^M\right]_{(F\times F)} \\ \vdots \\ \left[\hat{M}_D\right]_{(\lambda\times F)} * \left[B_l^M\right]_{(F\times F)} \end{bmatrix} \tag{3.10}$$

A network magnitude vector $\{\vec{\Phi}\}_{(F\times 1)}$ is formed by the magnitude of each screw motion ϕ as represented in equation 3.8.

The mechanical system is as established by the motion network matrix $[M_N]_{(\lambda.l\times F)}$, it implies that there are no motions trapped inside the loops, as the *circuit law* mentions for the sum of all the twists in a loop. The network unit motion matrix $[\hat{M}_N]_{(\lambda.l\times F)}$ is multiplied by the network magnitude vector of the motion system of screws $\{\vec{\Phi}\}_{(F\times 1)}$, as is represented in equation 3.11.

$$\sum \$^M \equiv [M_N]_{(\lambda \times F)} = [\hat{M}_N]_{(\lambda,F)}\{\vec{\Phi}\}_{(F\times 1)} = \{\vec{0}\}_{(\lambda\times 1)} \tag{3.11}$$

The result is equal to a zero vector $\{\vec{0}\}$, since there are no motions trapped inside the loops like in the circuit law, which mentions that the sum of all the twists in a loop will be zero (DAVIES, 2000).

The network motion matrix $[M_N]_{(\lambda.l\times F)}$ is a set of equations that determine the state of constraint of the mechanism. The lines of the matrix represents Plücker coordinates ordered by sets depending to the loop they correspond. The Plücker coordinates represent each one of the six independent degrees of freedom as explained in (DAVIES, 2000). For any closed loop, the relative motion between jointed elements should be zero (CAZANGI, 2008) so the circuit law presented by Davies (1981) as an analogy to Kirchoff tension law is satisfied as equation 3.12 shows. If the mechanical system is considered to have λ=6 in terms of screws it would be the result of the unitary motions of the direct couplings and their magnitude as expressed in equation 3.11

$$\sum L = \sum M = \sum N = \sum P = \sum Q = \sum R = 0 \tag{3.12}$$

The columns of $[M_N]_{(\lambda.l\times F)}$ represent each of the independent *dof* allowed by a kinematic pair in the mechanism. It is possible to determine if the equation system is linearly dependent by obtaining the rank m. If the rank is less than the number of rows in the unit motion matrix network $[\hat{M}_N]_{(\lambda l,F)}$, the system is linearly dependent. The rank in a matrix represents the number of linearly independent columns and it can be obtained using the Row echelon form or the Gauss elimination method. This overconstraint degree of the kinematic chain is obtained by the difference between the number of lines and m, as shown in equation 3.13. The net degree of freedoms is represented by (F_N) from the relationship between the independent equations, the rank (m) and the total of freedoms F. F_N indicates how many equations (primary variables) are required to determine the motion of the kinematic chain, this is shown in equation 3.14. The number of primary variables will be used in section 4.2.

$$C_N = \lambda l - m \tag{3.13}$$

$$F_N = F - m \tag{3.14}$$

3.2.2 Statics

A static analysis can be realized to determine the actions that influence a rigid body. With this analysis it is possible to determine the forces and torques that are being applied to a rigid body, determining the static equilibrium of a mechanism. Static analysis is done using the same frame of the mechanism as in instantaneous kinematics.

A representation of the actions in the mechanism can be visualized with the graph of actions G_A. The graph G_A is obtained from graph G_C by expanding each edge with more than one *doc* into parallel edges, as many as there are constraints in the kinematic pair. In this way the graph G_A will represent all the C constraints of all the c_i joints involved in the mechanism, this is shown in equation 3.15 where c_i is represented individually by each edge of the graph. Figure 16 represents the parallel expansion of a joint with tree constraints.

$$C = \sum c_i \tag{3.15}$$

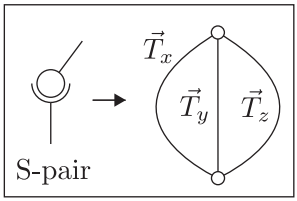


Figure 16 – Graph representation for the actions in an S-pair.

Graph G_A is a subgraph based on G_C where the expanded branches will continue being branches and the expanded chords will continue being chords. A cutset matrix $[Q]_{(k,e)}$ is related to the graph G_C, where, for the branches of the tree cut (k), each cut should pass through one branch and at least one chord. The positive orientation of the cut will be the same as the orientation of the branch that is being cut. For the graph G_A, the cutset matrix is $[Q^A]_{(k,C)}$. Equation 3.16 relates in its columns, all the c constraints in the mechanism, in the lines the matrix indicate if the constrained freedom is included in the cut set.

$$[Q^A]_{(k\times C)} = \begin{bmatrix} Q^A_{i,1} & Q^A_{i,2} & \cdots & Q^A_{i,C} \end{bmatrix}_{(1,C)} \tag{3.16}$$

A relationship among all the couplings can be identified using

the action matrix $[A_D]_{(\lambda,C)}$. The columns indicate the wrenches that represent the constraints c_i on the mechanism, the number of lines indicate the order of the screw system. Equation 3.17 represents how the unitary action matrix screws are arranged in the matrix.

$$[\hat{A}_D]_{(\lambda,C)} = \begin{bmatrix} \hat{\$}_a^A & \hat{\$}_b^A & \cdots & \hat{\$}_F^A \end{bmatrix} \tag{3.17}$$

The magnitude vector for the screw system is defined as $\{\vec{\Psi}\}_{(C\times 1)}$ and it contains the greatness of each wrench is shown in equation 3.18

$$\{\vec{\Psi}\}_{(C\times 1)} = \begin{Bmatrix} \psi_a \\ \psi_b \\ \vdots \\ \psi_F \end{Bmatrix} \tag{3.18}$$

The actions constraining a mechanism are related by the network action matrix $[A_N]_{(\lambda.k\times C)}$, which relates the cutsets from 1 to k and the identified wrenches from 1 to C. The unit network action matrix is formed by obtaining the diagonal of each line of the cutset matrix $[Q^A]_{(k\times C)}$ and then multiplying it by the unit action matrix $[\hat{A}_D]_{(\lambda\times C)}$. Equation 3.19 shows the diagonal for the line i of the cutset matrix.

$$[Q_i^A]_{(C,C)} = diag[Q_i^A]_{(1,C)} = [Q_i^A]_{(C,C)} \tag{3.19}$$

Equation 3.20 shows how the unitary action network matrix $[\hat{A}_N]$ is created, it requires the action matrix and the diagonal of the lines of the cutset matrix.

$$[\hat{A}_N]_{(\lambda.l\times F)} = \begin{bmatrix} \left[\hat{A}_D\right]_{(\lambda\times C)} * \left[Q_1^A\right]_{(C\times C)} \\ \left[\hat{A}_D\right]_{(\lambda\times C)} * \left[Q_2^A\right]_{(C\times C)} \\ \vdots \\ \left[\hat{A}_D\right]_{(\lambda\times C)} * \left[Q_l^A\right]_{(C\times C)} \end{bmatrix} \tag{3.20}$$

Notice that it is constructed in the same way as the unitary motion network matrix $[\hat{M}_N]_{(\lambda.l\times F)}$ in kinematics in equation 3.10. The action network is a set of equations where the unit action network and the magnitude vector are equal to a zero vector, as shown in equation 3.21.

$$\sum \$^A \equiv [A_N] = [\hat{A}_N]_{(\lambda.k \times C)} \{\vec{\Psi}\}_{(C \times 1)} = \{\vec{0}\}_{(\lambda.k \times 1)} \tag{3.21}$$

Each line of the action network matrix represents a Plücker coordinate, they are arranged depending on the loop they belong to. In the action unit the sum of all the wrenches that belong to the same cut will be zero, as the cutset law says in (DAVIES, 2000). This means that there are no actions trapped, causing the system to be overconstrained, this relationship is shown in equation 3.22 if the order of the screw system is $\lambda = 6$.

$$\sum L = \sum M = \sum N = \sum P = \sum Q = \sum R = 0 \tag{3.22}$$

In matrix $[\hat{A}_N]_{(\lambda k, C)}$ the rank (a) indicates the number of independent equations the system has. To obtain the number of net freedoms F_N of the constraint system is indicated in equation 3.23. The C_N (primary variables) indicates the number of equations that have to be determined in equation 3.24, normally they are related to the actuators or to any external force that activates the mechanism.

$$F_N = \lambda k - a \tag{3.23}$$

$$C_N = C - a \tag{3.24}$$

The obtained parameters F_N and C_N are interpreted using Figure 17. The figure separates the rigid structures with $F_N = 0$ from the kinematic chains with $F_N > 0$, the figure also makes a distinction in the rigid or kinematic structures indicating where $C_N > 0$ that the structure is overconstrained and for structures with $C_N = 0$ that they are not overconstrained.

3.3 PATH METHOD

The path method is a methodology for instantaneous kinematic analysis for assembly of a mechanism using Screw Theory (SHUKLA; WHITNEY, 2005). The method determines the state of constraint using twists $\$^M$ and wrenches $\$^A$. The twists can be united in a *union* matrix as shown in equation 3.25. The screws can also be intersected by finding the elements that the set of screws has in common. Equation 3.26

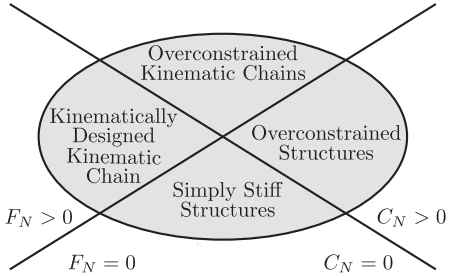


Figure 17 – Closed Passive coupling networks, (DAVIES, 2000).

represents the intersection.

$$\text{Union } (\$_1, \$_2, \cdots, \$_n) = \begin{bmatrix} \$_1 \\ \$_2 \\ \cdots \\ \$_n \end{bmatrix} \tag{3.25}$$

$$\text{Intersection } (\$_i) = Reciprocal \begin{bmatrix} Reciprocal(\$_1) \\ Reciprocal(\$_2) \\ \cdots \\ Reciprocal(\$_n) \end{bmatrix} \tag{3.26}$$

The method considers all the possible paths from one pairing element to another, covering all the pairing elements and joints of the mechanism. The analysis is limited to mechanisms without cross-coupling, this is when a dependent *dof* is identified in the mechanism. The dependency makes the algorithm deliver incorrect answers and suggests that a more complex methodology like the Davies method is best suited to analyze the mechanism (SHUKLA; WHITNEY, 2005). Figure 18 illustrates a five link mechanism with the possible paths from a fixed element to an output element. In a) the only way for the paths to travel through the links and joints while having the bottom element fixed creates crossed paths. A possible solution would be changing the fixing element to be the pairing element at the left, as shown in b). This change makes it possible to analyze the mechanism with this methodology.

The methodology represents mechanisms using graphs. The used graphs identify the pairing elements and the kinematic pairs with nodes, edges without representation are used to connect the nodes. Figure 19,

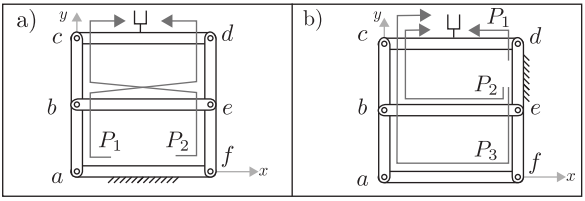


Figure 18 – a) Mechanism with cross paths, the fixed pairing element is at the bottom, in b) the fixed element changed is the left pairing element, eliminating the cross paths.

in a) presents how two pairing elements (black nodes) are connected by a kinematic pair (white node). If the kinematic pairs are arranged in a serial formation then they can be united, b) provides a graph representation for the union in equation 3.25. For the intersection of the screws in equation 3.26, the graph representation is shown in part c) of the same figure, this is applied for kinematic pairs that form a parallel arrangement between two pairing elements.

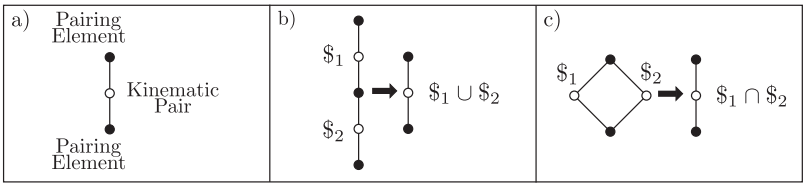


Figure 19 – Graph representation for the Path Method: a) two pairing elements connected by a kinematic pair, b) union operation for two joints in series, c) intersection operation for two joints in parallel.

A path unites the screws of the kinematic pairs where it passes. Equation 3.27 shows the screws for the paths in Figure 18, each path represents the motion of the links where the path passes. Note, the method of arranging the screws is transposed from how they were presented in section 2.2.2.

$$\begin{aligned} [M_1] &= [\$_d]^T \\ [M_2] &= [\$_e \$_b \$_c]^T \\ [M_3] &= [\$_e \$_f \$_a \$_b \$_c]^T \end{aligned} \tag{3.27}$$

To perform the analysis the paths are identified. Since $[M_1]$, $[M_2]$ and $[M_3]$ are paths of the same mechanism, a reciprocal matrix for each path is obtained in equation 3.28. This matrix corresponds to the actions of the links of the corresponding path. An implemented algorithm, based on the null space can be found in (WHITNEY, 2004). The action matrices are united into a single matrix as shown in equation 3.29. Then again, the reciprocal of such a matrix is obtained as shown in equation 3.30 to determine the mobility of the closed loop mechanism. The result for this mechanism shows motion along the x axis.

$$\begin{aligned} [A_1] &= recip([M_1]) \\ [A_2] &= recip([M_2]) \\ [A_3] &= recip([M_3]) \end{aligned} \tag{3.28}$$

$$[A_u] = \begin{bmatrix} [A_1] \\ [A_2] \\ [A_3] \end{bmatrix} \tag{3.29}$$

$$[M_{123}] = recip([A_u]) \tag{3.30}$$

The constraint analysis will determine if a mechanism is overconstrained. The method obtains the action matrices by finding the reciprocal of the screws in the first two paths in equation 3.31. The first two action matrices are grouped in a union matrix, shown in equation 3.32, the reciprocal of the union matrix identifies the mobility of both paths in equation 3.33.

$$\begin{aligned} [A_1] &= recip([M_1]) \\ [A_2] &= recip([M_2]) \end{aligned} \tag{3.31}$$

$$[A_{u12}] = \begin{bmatrix} [A_1] \\ [A_2] \end{bmatrix} \tag{3.32}$$

$$[M_{12}] = recip([A_{u12}]) \tag{3.33}$$

To analyze the overconstraints in the mechanism, the third path is grouped with the motion results of the first two paths, as equation 3.34 shows, which is the same result as equation 3.30. The intersection of the screws indicate the actions of the linked system that are constraining the mechanism as equation 3.35 shows. This constraint analysis indicates that the output element is overconstrained in the x and in z axes, therefore the mechanism is overconstrained.

$$[M_{123}] = \begin{bmatrix} [M_{12}] \\ [M_3] \end{bmatrix} \tag{3.34}$$

$$[A_{123}] = recip([M_{123}]) \tag{3.35}$$

For more complex mechanisms, the methodology is the same, incorporating the motions and action one at a time, until all of them are analyzed thus determining the state of the mechanism.

3.4 CHAPTER CONCLUSIONS

This chapter has presented three different methodologies for analyzing mechanisms. The table method presents a quick way to analyze mechanisms, nevertheless the complexity of the methodology is related to the number of loops, since many possible solutions can be found when replacing missing linear freedoms. The path method is based on the reciprocity, union and intersection of the screws to determine the mobility and the overconstraints in a mechanism. Davies presents a methodology where it is possible to determine any linear dependency in the mechanical system, in this way making it possible to determine if any motion component in the mechanism is overconstrained. The Davies method is used in the next chapter, the network unit motion matrix and the unitary action network matrix are used in a freedom and constraint dependency analysis. Both matrices can be obtained using the functions in appendix E, F, G and H.

4 ANALYSIS

This chapter is dedicated to the explanation of how this research determines linear dependent freedoms and constraints in a mechanism. Using the Davies method it is possible to obtain a pair of matrices that determine the motion network and the action network of a mechanism. The matrices $[\hat{M}_N]_{(\lambda.l\times F)}$ and $[\hat{A}_N]_{(\lambda.k\times C)}$ are used to know which are the dependent equations that constrain a mechanism. It is important to note that each matrix represents two equation systems, the columns represent the twists or the wrenches. Depending if it is a motion analysis or a static analysis, this will determine the velocities and the actions that describe relative motion between the pairing elements of a mechanism. The other equation system relates the Plücker coordinates that describe the six components of motion with the identified loops.

This chapter is divided into two sections, section 4.1 explains what a linear dependent vector is and how the rank represents independent vectors in the system. Section 4.2 outlines the algorithm search for a dependent and independent sets of vectors. The results of the algorithm applied to a slider-crank mechanism are discussed at the end of the section, together with some of the limitations of the algorithm.

4.1 LINEAR INDEPENDENCE AND RANK

When solving simultaneous linear equations, dependent elements may appear if the system has more unknowns than equations. The system of equations in 4.1 represents m equations and n variables. The coefficients a_{ij} and the quantities of b_i are known in the equation system, the unknowns are the values of the x_j components, where $i = 1, 2, \cdots, m$ and $j = 1, 2, \cdots, n$. If $m < n$, the system has more unknowns than equations (BRONSON, 1991).

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n &= b_1, \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n &= b_2, \\ &\vdots \\ a_{m1}x_1 + a_{m2}x_2 + \cdots + a_{mn}x_n &= b_m. \end{aligned} \tag{4.1}$$

The equation system may be represented by the matrix form as

equation 4.3 shows, and the system is called homogeneous if $b = 0$ (LEDUC, 1996). This same matrix form can be observed in equations 3.11 and 3.21. A solution to the equation system can possibly be archived using Gaussian Elimination for solving simultaneous linear equations.

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix}, \quad \mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}, \quad \mathbf{b} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix}. \tag{4.2}$$

$$\mathbf{Ax} = \mathbf{b} \tag{4.3}$$

In (BRONSON, 1991) equations 4.4 and 4.5 are given as definitions of a linear independent and dependent vector. A vector $\mathbf{V}_1$ is a linear combination of other vectors $\mathbf{V}_2$, $\mathbf{V}_3$,$\cdots$, $\mathbf{V}_n$ if there exist d scalars that satisfy equation 4.4.

$$\mathbf{V}_1 = d_2\mathbf{V}_2 + d_3\mathbf{V}_3 + \cdots + d_n\mathbf{V}_n. \tag{4.4}$$

A set of vectors of the same order $\{\mathbf{V}_1, \mathbf{V}_2, \cdots, \mathbf{V}_n\}$ are linearly dependent if there exist scalars c, not all zero, that satisfy equation 4.5. For the vectors to be linearly independent the only set of scalars that satisfy such an equation is $c_1 = c_2 = \cdots = c_n = 0$ (POOLE, 2006). This means no dependent vectors were found in the set. If linearly dependent vectors are found then the solution will be a particular non zero solution for the scalars c.

$$c_1\mathbf{V}_1 + c_2\mathbf{V}_2 + c_3\mathbf{V}_3 + \cdots + c_n\mathbf{V}_n = \mathbf{0}. \tag{4.5}$$

The number of independent vectors in a set is considered to be the rank of the matrix which can be obtained using row reduction or singular value decomposition like GNU Octave does. In (MUROTA, 2000) different algorithms for the rank are described. The rank represents the number of linearly independent rows and linearly independent columns with the same number, therefore the column rank and row rank of the same matrix are equal (LEDUC, 1996).

For example, set $\mathbf{A}$ is a set of vectors shown in 4.6, the components of the vectors are shown in 4.7,and the matrix form in 4.8 has a rank of two, which means that two dependent equations are found in the matrix. The row reduced form shown at the right in 4.8,is obtained by (BRONSON, 1991) when the author presents the Gaussian elimina-

tion. Notice the two rows with zeros, they explain the two dependencies also known as primary variables.

$$A = \left\{ \begin{array}{cccc} \mathbf{V}_1, & \mathbf{V}_2, & \mathbf{V}_3 \end{array} \right\}, \tag{4.6}$$

$$\mathbf{V}_1 = \begin{bmatrix} 1 \\ 2 \\ 3 \\ 5 \end{bmatrix}, \quad \mathbf{V}_2 = \begin{bmatrix} 3 \\ -1 \\ 2 \\ 15 \end{bmatrix}, \quad \mathbf{V}_3 = \begin{bmatrix} 4 \\ 1 \\ 5 \\ 20 \end{bmatrix} \tag{4.7}$$

$$A = \begin{bmatrix} 1 & 3 & 4 \\ 2 & -1 & 1 \\ 3 & 2 & 5 \\ 5 & 15 & 20 \end{bmatrix} \quad \rightarrow \begin{bmatrix} 1 & 3 & 4 \\ 0 & 1 & 1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \tag{4.8}$$

4.2 ALGORITHM

This section describes the sequence of steps proposed by this research to find linear dependent equations in a system of motions and in a system of actions of a mechanism. To reach the objective the algorithm is separated into three main steps. The figures shown in this section illustrate the sequence of partial results needed to achieve the goal.

The first step, using the Davies method, is to set up the matrices that describe the freedoms and the constraints in a mechanism, for this, twists and wrenches will be used. Figure 20 illustrates the inputs required by the algorithm to perform the kinematic analysis. Vectors $\vec{S}^M$, $\vec{S}_0$, h_m and the circuit matrix $[B^M]$ are used for the motion analysis and vectors $\vec{S}^A$, $\vec{S}_0$, h_a and the cutset matrix $[Q^A]$ for static analysis, all of these matrices are inputs to the algorithm. This information is read by implemented functions found in the appendix E, F, G and H of this study, which results in sets of linearly dependent and independent screws. The workflow goes as follow: first the screw matrices are obtained so they can be combined according to the Davies method to determine the action network matrix and the motion network matrix, as well as their rank, the net freedoms and the net constraints for both systems. At the end, the value of the net freedom and the net constraint describe the state of constraint of the mechanism.

For the the second step, the motion and action network matrices are separated into blocks identified in both matrices. These blocks are found using the decomposition of Dulmage and Mendelsohn (DM), see

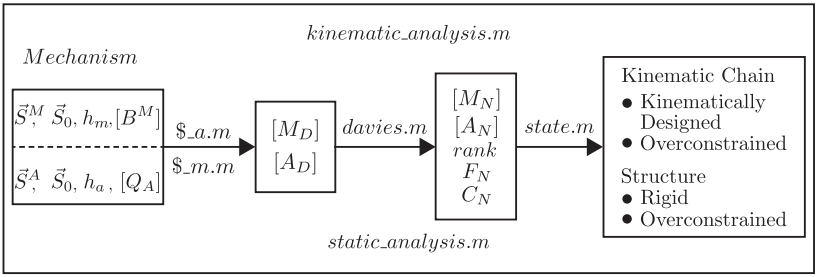


Figure 20 – Diagram representing the first step.

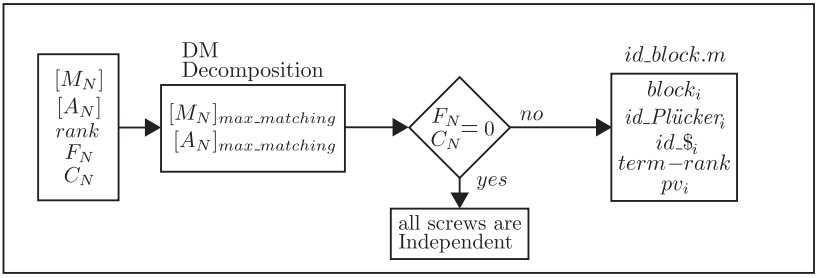


Figure 21 – Diagram representing the second step.

appendix C. The decomposition returns a maximum matching matrix, one for kinematics and another for statics. If the number of primary variables F_N or C_N is zero the algorithm will consider the screws of $[M_N]$ and $[A_N]$ to be independent, otherwise the screws are not all independent. The blocks found in DM are separated from the maximum matching matrix. Those blocks where all the elements of the matrix are zero will not be considered, leaving only n blocks to be analyze. Each $block_i$, where $i = 1, 2, \cdots, n$, has its own $term - rank$, this parameter leads to a particular number of primary variables of the block pv_i. Each block is identified with auxiliary vectors, for the screws $id\_\$_i$ and $id\_Plücker_i$ for the coordinates that compose such a block. Figure 21 illustrates the process that is performed by the algorithm.

The third step aims to obtain all possible combinations of screws in $block_i$. This step is performed to identify the dependencies indicated by the number of primary variables (pv_i). First, the algorithm checks for columns in $block_i$ where all elements are zero. A single column with all zero elements will drop the number of primary variables in one unit. In this way, if the number of columns with all zero elements is the same

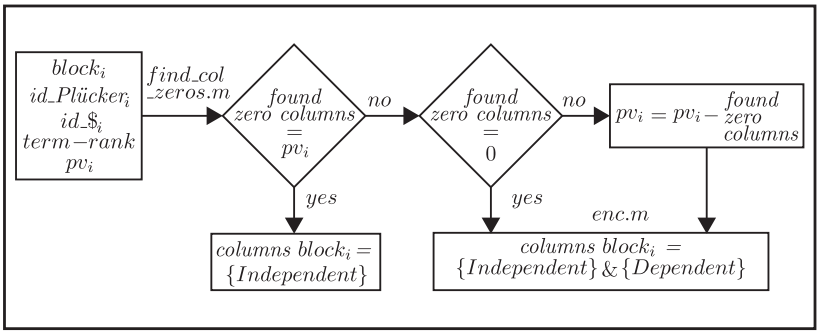


Figure 22 – Diagram representing the third step.

as the number of pv_i then the remaining columns without all zero elements are considered to be a set of independent screws in $block_i$. In case $pv_i \geq 1$, $block_i$ may be without columns with all zero elements. Now, the algorithm will systematically remove sets of columns to compare the term-rank of $block_i$ with the term-rank of the sub-matrix formed by the set of remaining columns. The algorithm will remove one column from left to right , and then two columns at a time. It will remove columns and it will verify if the rank decreases, in such a case the removed columns are considered to be a set of linearly independent columns of $block_i$. If the rank remains the same such removed columns are then a set of linearly dependent columns of $block_i$. Figure 22 shows the conditions that lead to find linear dependent and independent sets of screws from $block_i$. For this step, all considered blocks from the maximum matching matrix will return an array containing sets of combinations of screws classified as either linearly dependent or independent. The functions shown in appendix K and L return the sets found in each block.

One can find the main functions of the algorithm in appendix I and J. They receive the description of the mechanism as described at the first step and perform the analysis by calling other functions included inside them that will act according to what is described in the second and in the third step.

To exemplify the algorithm described above, a slider-crank mechanism is analyzed. The algorithm is set to be executed twice, each time or case it analyzes a different joint configuration of the same mechanism. In this way the difference between an overconstraint and a kinematically designed state of the same mechanism is identified. The change of

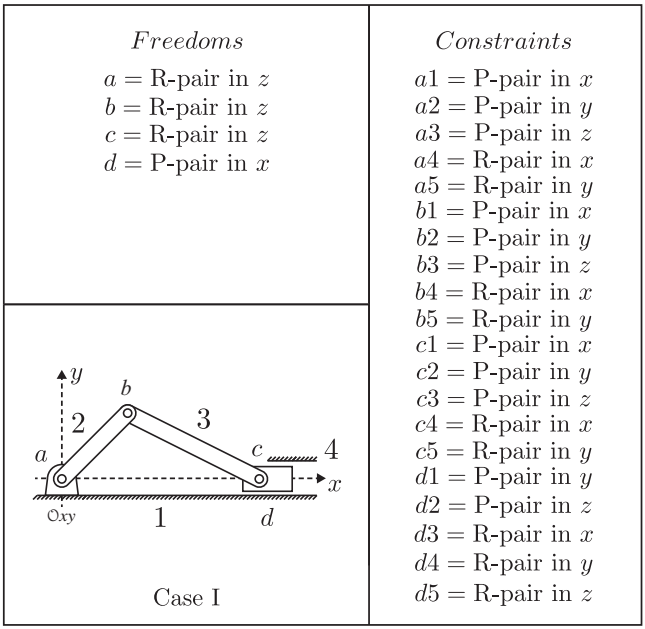


Figure 23 – Case I: Freedoms and constraints for a slider-crank mechanism.

the kinematic pairs is realized according to (RESHETOV, 1979) where a list of rational structures of the slider-crank mechanism is shown as possible solutions that eliminate overconstraint. This same solution for the motion analysis was analyzed in section 3.1 in the table method.

The example illustrated in Figure 23 shows the slider-crank mechanism in a two dimensional space, the pairing elements are numbered from one to four and the kinematic pairs are identified with letters. The tables indicate both the total freedoms and constraints that each kinematic pair has. Due to considerations of space, all action matrices may be found in appendix D for Case I and Case II.

The matrices in 4.9 and D.1, shows the motion network matrix and the action network matrix for the first case of the slider-crank mechanism with three R-pairs and one P-pair. Each screw is represented with a letter matching the joint they describe, the lines of the matrix are identified with the Plücker motion coordinates.

$$[M_N] = \begin{array}{c} \\ L \\ M \\ N \\ P \\ Q \\ R \end{array} \begin{array}{c} \begin{array}{cccc} a & b & c & d \end{array} \\ \left[\begin{array}{cccc} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 \\ 0 & -2 & 0 & -1 \\ 0 & 2 & 6 & 0 \\ 0 & 0 & 0 & 0 \end{array}\right]_{(6,4)} \end{array} \tag{4.9}$$

The rank of the motion network matrix is $m = 3$ and the number of gross freedoms is $F = 4$. The number of primary variables $F_N = 1$ and $C_N = 3$ indicate the state of constraint of the mechanism according to Figure 17. It was determined that the slider-crank mechanism with these configuration of joints is an overconstrained kinematic chain. The rank of the action network matrix is $a = 16$, the number of gross constraints is $C = 20$. The number of primary variables $F_N = 2$, indicating the number of dependent equations that may be removed, while $C_N = 4$ is related to the overconstraint network indicating the external actions internalized in the mechanism.

$$[M_N]_{mxm} = \begin{array}{c} \\ N \\ P \\ Q \\ \hline L \\ M \\ R \end{array} \begin{array}{c} \begin{array}{cccc} d & a & b & c \end{array} \\ \left[\begin{array}{cccc} 0 & 1 & 1 & 1 \\ -1 & 0 & -2 & 0 \\ 0 & 0 & 2 & 6 \\ \hline 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{array}\right] \end{array} \tag{4.10}$$

The matrices in 4.10 and D.2 show the maximum matching of $[M_N]$ and $[A_N]$, both are in block triangular form. These matrices and their blocks result from Dulmage-Mendelsohn Decomposition. Two blocks are found in $[M_N]$, one of them has all the elements zero so the algorithm will not consider it, the considered block will be only the upper part of the matrix. For $[A_N]$ twenty five blocks have been identified, only six of them are considered for the analysis due to not being blocks with all zero elements.

$$Dependent = \begin{bmatrix} \{d\}\{a\}\{b\}\{c\} \\ 0 \end{bmatrix} \tag{4.11}$$

$$Independent = \begin{bmatrix} \{d,a\}\{d,b\}\{d,c\}\{a,b\}\{a,c\}\{b,c\} \\ 0 \end{bmatrix} \tag{4.12}$$

In 4.11 the matrix shows the identified dependent set of vectors, and in 4.12 the matrix shows those sets of independent screws for the $[M_N]_{mxm}$ for Case I, separated by each block found using DM Decomposition. These results indicate how the freedoms in the mechanism are related to one another. In D.3 and D.4 the identified sets for the linearly dependent and independent screws in the action matrix network $[A_N]_{mxm}$ are shown. For the motion analysis of the slider-crank mechanism in Case I, dependent sets found indicate that all four freedoms are dependent on each other in a closed loop, meaning that when all joints are connected they depend on each other to be capable of allowing motion while connected. On the other hand, when examining pairs of joints we find independence of motion, since there are no constraints that restrain motion in the direction allowed by those two joints that are being analysed. In other words, if the three bodies are connected they can have independent motion allowed by the two joints found as independent, and if the four bodies are connected to each other, then a joint depends on the other three to be capable of moving.

Matrix 4.13, shows the motion matrix for the second case in Figure 24. It shows the description of the joints used as a solution to eliminate overconstraint. The number of gross freedoms is $F = 7$ and the rank of the matrix is $m = 6$, the net freedom is $F_N = 1$ and the net constraint is $C_N = 0$. This result indicates that the mechanism is kinematically designed according to Figure 17. For the static analysis matrix D.5 shows the action matrix for this second case. The number of gross constraints is $C = 17$, the rank of the matrix is $a = 3$. The number of primary variables is $F_N = 3$ and $C_N = 2$ relating the overconstraints in the state of actions of the mechanism.

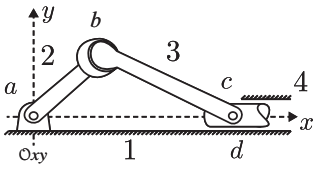
Freedoms	*Constraints*
a = R-pair in z $b1$ = R-pair in x $b2$ = R-pair in y $b3$ = R-pair in z c = R-pair in z $d1$ = P-pair in x $d2$ = R-pair in x	$a1$ = P-pair in x $a2$ = P-pair in y $a3$ = P-pair in z $a4$ = R-pair in x $a5$ = R-pair in y $b1$ = P-pair in x $b2$ = P-pair in y $b3$ = P-pair in z $c1$ = P-pair in x $c2$ = P-pair in y $c3$ = P-pair in z $c4$ = R-pair in x $c5$ = R-pair in y $d1$ = P-pair in y $d2$ = P-pair in z $d3$ = R-pair in y $d4$ = R-pair in z
 Case II	

Figure 24 – Case II: Freedoms and constraints for a slider-crank mechanism.

$$[M_N] = \begin{array}{c} \\ L \\ M \\ N \\ P \\ Q \\ R \end{array} \begin{array}{c} \begin{array}{ccccccc} a & b1 & b2 & b3 & c & d1 & d2 \end{array} \\ \left[\begin{array}{ccccccc} 0 & 1 & 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & -2 & 0 & 0 & -1 \\ 0 & 0 & 0 & 2 & 6 & 0 & 0 \\ 0 & 2 & -2 & 0 & 0 & 0 & 0 \end{array}\right]_{(6,7)} \end{array} \tag{4.13}$$

The maximum matching matrix for $[M_N]$ is shown in 4.14, the matrix shows sixteen blocks identified by Dulmage-Mendelsohn decomposition, only six will be considered by the algorithm for the linear analysis. The maximum matching matrix for $[A_N]$ is shown in D.6, it has one hundred blocks returned from DM decomposition, only fifteen of them are blocks with not all elements zero.

$$[M_N]_{mxm} = \begin{array}{c} \\ N \\ P \\ Q \\ L \\ R \\ M \end{array} \left[\begin{array}{c|cccc|ccc} & d2 & a & b3 & c & d1 & b1 & b2 \\ \hline & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ & -1 & 0 & -2 & 0 & 0 & 0 & 0 \\ & 0 & 0 & 2 & 6 & 0 & 0 & 0 \\ \hline & 0 & 0 & 0 & 0 & -1 & 1 & 0 \\ \hline & 0 & 0 & 0 & 0 & 0 & 2 & -2 \\ \hline & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{array}\right] \tag{4.14}$$

As mentioned before, the output of the algorithm is: a linearly dependent set and a linearly independent set found in each block. For the second case of the slider-crank mechanism illustrated in Figure 24, the identified independent sets for the motion analysis are shown, in 4.15 for the dependent sets and 4.16 for the independent sets found in $[M_N]$. The static analysis results of linearly dependent and independent vectors can be found in appendix D.2

$$Dependent = \left[\begin{array}{c|c|c|c} \{d2\}\{a\}\{b3\}\{c\} & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 \end{array}\right] \tag{4.15}$$

$$Independent = \left[\begin{array}{c|c|c|c} \{d2,a\}\{d2,b3\} \\ \{d2,c\}\{a,b1\} \\ \{a,c\}\{b3,c\} & 0 & 0 & 0 \\ \hline 0 & \{d1\} & \{b1\} & 0 \\ \hline 0 & 0 & \{b1\} & \{b2\} \\ \hline 0 & 0 & 0 & \{b2\} \end{array}\right] \tag{4.16}$$

For the slider-crank mechanism used in this example, the result for the kinematic equations of the first case indicate all the individual screws as linearly dependent between each other, and the linearly independent sets indicate the possible combinations between two screws. The interpretation of such results imply a redundancy of constraint in the kinematic chain caused by a close dependent relationship between the equations of all four screws. To eliminate redundancy and close dependency between the screws, Case II presents the same mechanism with a different joint configuration as a possible solution for such over-constraint, as mentioned above. The screws of joints $d2, a, b3$ and c are linearly dependent between each other, while joints $d1, b1$ and $b2$

are independent. The results for the independent sets indicate those freedoms added to the joints in this case, two extra rotations permitted by the S-pair, and rotation allowed by the C-pair, allowing pairing element 3 to be capable of rotating in x and y axes and eliminating misalignment in the plane xy.

The static analysis for Case I shows a close dependent relation between the action screws. For the independent sets, nineteen pairs of screws were found independent in the first block, with three more single screws $a1, b1, c1$ and $d5$ in the other analyzed blocks. The single independent screws represent the moment vector in the x axis and a force vector in the z axis, which represents the force and moment directions that are active in the mechanism. For Case II the dependent sets of vectors show a close relationship between the joints $d1, d2, a2, a3, b3, b3, c3$ and $c3$ that intimately relates force acting in the y and z axes, while the independent sets show joints $a1, a4, a5, b1, c1, c4, c5, d3$ and $d4$ as being independent equations in the action matrix.

The algorithm presents a linear analysis performed on each block found by the decomposition of DM. It is observed that extra freedoms allowed by the joints that are used for the elimination of redundant constraints are considered as independent in the motion analysis. The reduction of action screws in the static analysis makes it possible to identify less dependent screws in each block while the independent screws augment.

To summarise this section, a brief explanation on how to use the proposed algorithm will be reviewed. First, for a given mechanism, the number of bodies and the type of joints with their degrees of freedom will need to be identified in a two or three dimensional space depending on the type of mechanism. This is done to obtain the position vector $\vec{S}_0$ from the origin of the coordinate system to the actual position of the joint. Second, it is recommended to obtain the coupling graph G_C and identify those joints that posses more than one degree of freedom, in order to generate the motion graph G_M and the action graph G_A. These two graphs will help identify each screw, its direction vector $\vec{S}$ and its pitch h. From each graph the circuit matrix $[B^M]$ and cutset matrix $[Q^A]$ may also be obtained. Third, a position matrix and a matrix with the direction of the screws will be formed by each position vector and each screw direction. The vectors must be arranged by columns that match to the joint they represent. The pitches of each joint will form a binary line vector, where 0 indicates if the joint is rotational and 1 if the joint is translational. Once these matrices are obtained from the the original mechanism, then the functions of *kinematic_analysis*

and *static_analysis* can be used, see appendix I and J. In appendix M the inputs for Case I and Case II are shown as an example of how the input matrices should look before they can be used. Finally GNU Octave may be used to process this information in order to determinine the dependency analysis.

For future implementations of the algorithm, linear dependencies should be automatically analyzed for further relationships between the elements of a set and the elements of other sets in the same block, this is one limitation of the algorithm. Another limitation of this algorithm is the programing language used, since it is limited by the number of possible combinations between columns in a matrix. The time of response of the example presented above was 0.4 seconds, for the two loop mechanism in figure 14 the algorithm takes 1.04 seconds for the kinematic analysis and 3.05 seconds for the static analysis. The algorithm as presented in this research is set to find only pairs of linearly dependent and independent screws. A better and more representative way to determine dependency and independency would consider analyzing all combinations possible between the screws but attempting this consumed too much time, making it difficult to determine the result of mechanisms with high complexity.

5 CONCLUSIONS

This dissertation proposes an algorithm for kinematic analysis of mechanisms. Based on the freedoms and constraints allowed by the joints it is possible to determine the state of constraint of such mechanisms. For this, linearly dependent and independent sets of screws are found to identify the redundant constraints in a mechanism. Using kinematics as presented by (HUNT, 1978), relative motion between the rigid bodies is studied. The intention of this research is to perform a kinematic analysis of a mechanism. As a result of a proposed algorithm for kinematic analysis, an extensive list of sets of dependent and independent screws are given for further examination and interpretation. The identification of common dependent screws in the sets may lead an engineer to consider exchanging certain kinematic pairs for others that allow more degrees of freedom and in this way eliminate any presented overconstraints. This exchange process will depend on the expertise of the engineer to identify which determined joints are exchange capable.

In chapter 2, this text presented a review of the concepts of freedom and constraint used in the Theory of Mechanisms as a way to describe motion. This was followed by a discussion of the kinematic principle where six independent motion components mentioned in (POLLARD, 1933) are presented. In that same chapter, the kinematic pairs are described as the link between the rigid bodies, allowing certain freedoms that allow relative motion between the pairing elements. A mechanism can be represented by a kinematic chain where the rigid bodies are represented and connected by joints (TSAI, 2001). The mobility of a kinematic chain is presented as a parameter that indicates the number of independent coordinates that determine the configuration of a mechanism (IONESCU, 2003). Also in this chapter, Graph Theory by (CHRISTOFIDES, 1975) is used to represent mechanisms. Screw Theory based on (DAVIES, 2000) is used to describe the relative motion between the pairing elements of a mechanism. Both theories are used in the analysis, to determine vectors and matrixes, that relate the kinematic pairs with the pairing elements in a determined space where motion is allowed.

In the same chapter 2, a review of the literature of Constraint Design (CD) methodologies is presented. This review included a discussion of Exact CD mentioned by (KAMM, 1990) and (SKAKOON, 2009) which are used as methodologies for designing fixtures, rigid structures and mechanisms using exactly six constraints, one for each motion com-

ponent. Redundant CD is used to increase load capacity and stability by constraining a motion component in a redundant way. In (POLLARD, 1933) it is mentioned that if the constraints that define an element exceed six, force will be required to accomplish the seventh constraint. The opposite phenomenon is found in underconstrained mechanisms, where one or more of the six motion components are left without restriction. Such unconstrained freedom is considered to be a dangerous mobility, since the component is free to move (WHITNEY et al., 1999). The discussion on self-aligning at the end of this second chapter explains the benefits of both active and passive self-aligning. In (RESHETOV, 1979) self-aligning is achieved in overconstrained mechanisms by changing kinematic pairs for others with a greater degree of freedom.

Three different methods for kinematic analysis were explained and exemplified in chapter 3. The first method is taken from (RESHETOV, 1979) and (SZYDLOWSKI, 2000). The second method presented is the Davies Method from (DAVIES, 2000) and (DAVIES, 2006). It uses Graph Theory and Screw Theory to set static and motion equations that describe the mechanism. The state of constraint of a given mechanism will be defined by the number of primary variables found after the rank of the motion network matrix $[M_N]$ and in the action network matrix $[A_N]$. The third methodology described in that chapter is the Path Method from (SHUKLA; WHITNEY, 2005). It determines the state of constraint use by finding all the possible paths in the kinematic chain to go from one element assumed fixed to another one of interest. In this research the Davies Method is applied to determine the matrixes that will be analyzed using the algorithm proposed in this study.

The algorithm proposed by this research represents an analysis of the motion and action matrixes obtained from the Davies Method. Chapter 4 defines what a linearly dependent and independent vector is, and how a matrix, formed by vectors, may have a number of independent vectors represented by the range of the matrix. The proposed algorithm uses the Dulmage and Mendelsohn Decomposition implemented in GNU Octave, such decomposition is used to find a maximum matching bipartite graph which arranges the motion and action matrixes into a block superior triangular form. The presented algorithm analyses each block and returns all the independent and dependent sets of screws. The algorithm is presented as a three step procedure, described in this same chapter. The last part of the chapter indicates how to proceed in order to obtain the inputs required by the algorithm. The appendix section of this dissertation presents all the algorithms developed by this research, it also includes the input file for the example

Case I and Case II, used in chapter 4. Further theory on Screw Theory and the Dulmage and Mendelsohn decomposition may also be found in the appendix section.

Finally the results of the presented algorithm for a kinematic analysis of a mechanism contain the possible sets of screws separated by blocks found from the Dulmage and Mendelsohn decomposition. Although there may be a lot in the block, each block should be analysed with as many columns as it may have. The results indicate that for complex mechanisms, the increment of columns in the matrixes lead to difficulties, resulting in an increment of the time of response from Octave. Such complications with the used programming language may lead to possible future implementations where considering other programming languages to speed up the time of execution may be a solution. Future researchers may consider C++ as an option to eliminate this limitation. In this way all the combinations between the columns may be obtained without limiting it to only two. It is recommended that to improve the work here presented an algorithm that automatically identifies those dependent and independent sets may be implemented as well, to ease the identification process once the results are obtained from Octave.

REFERENCES

BALL, R. S. *A treatise on the theory of screws*. [S.l.]: Cambridge: Cambridge University Press, 1998.

BEDFORD, A. M.; FOWLER, W. *Engineering Mechanics Statics*. 5. ed. [S.l.]: Pearson Printice Hall, 2008.

BLANDING, D. K. *Exact Constraint: Machine Design Using Kinematic Principles*. New York, NY, USA: ASME Press, 1999.

BRONSON, R. *Matrix Methods An introduction*. [S.l.]: Academic Press, Inc., 1991.

BROWN, H. T. *Five Hundred and Seven Mechanical Movements*. [S.l.]: reprinted by The Astragal Press, 1995.

BUNUS, P.; FRITZSON, P. Methods for structural analysis and debugging of modelica models. In: *2nd International Modelica Conference, Proceedings*. [S.l.: s.n.], 2002. p. 157–165.

CAMPOS, A. A. *Cinemática Diferencial de Manipuladores Empregando Cadeias Virtuais*. Tese (Doutorado) — Universidade Federal de Santa Catarina, Março 2004.

CARBONI, A. P. *Análise Conceitual de Estruturas Cinemáticas Planas e Espaciais*. Tese (Doutorado) — Universidade Federal de Santa Catarina, Fevereiro 2008.

CAZANGI, H. R. *Aplicaçao do Método de Davies Para Análise Cinemática e Estática de Mecanismos Com Múltiplos Graus de Liberdade*. Dissertação (Mestrado) — Universidade Federal de Santa Catarina, 2008.

CHRISTOFIDES, N. *Graph Theory: An Algorithmic Approach*. [S.l.]: Academic Press, London., 1975.

DAI, J. S. et al. Mobility analysis of a complex structured ball base on mechanism decomposition and equivalent screw system analysis. *Mechanism and Machine Theory*, v. 39, p. 445–458, 2004.

DAVIES, T. H. Systematic pair-reduction procedure for constraint problems. *ASME publication 68-Mech 59. [Presented at the tenth Mechanisms Conference, 1968] New York: American Society of Mechanical Engineers*, 1968.

DAVIES, T. H. *Self-Aligning in Mechanisms*. [S.l.], 1970. v. 1.

DAVIES, T. H. Circuit actions attributable to active couplings. *Mechanism and Machine Theory*, v. 30, n. 7, p. 1001–1012, 1995.

DAVIES, T. H. Couplings, coupling network and their graphs. *Mechanism and Machine Theory*, v. 30, n. 7, p. 991–1000, 1995.

DAVIES, T. H. *The 1887 committee meets again. Subject: freedom and constraint*. [S.l.], July 2000. 56 p.

DAVIES, T. H. Freedom and constraint in coupling networks. *Proceedings of the Institution of Mechanical Engineers, Part C: Journal of Mechanical Engineering Science.*, v. 220, n. 7, p. 989–1010, 2006.

DOWNEY, K.; PARKINSON, A.; CHASE, K. An introduction to smart assemblies for robust design. *Research in Engineering Design*, v. 14, n. 4, p. 236–246, November 2003.

DULMAGE, A. L.; MENDELSOHN, N. S. Coverings of bipartite graphs. *Canadian Journal of Mathematics*, The Canadian Mathematical Society by The University of Toronto Press, v. 10, n. 4, p. 517–534, 1958.

GOGU, G. *Structural Synthesis of Parallel Robots, Part 1 - Methodology*. [S.l.]: Springer, 2008.

HALE, L. C.; SLOCUM, A. H. Optimal design techniques for kinematic couplings. *Journal of the International Societies for Precision Engineering and Nanotechnology*, v. 25, n. 2, p. 114–127, 2001.

HAMMOND, A. M. *Establishing a Quantitative Foundation for Exactly Constrained Design*. Tese (Doutorado) — Brigham Young University, April 2004.

HUNT, K. H. *Kinematic Geometry of Mechanisms*. [S.l.]: Oxford University Press, 1978.

IONESCU, T. G. Iftomm definitions. *Mechanism and Machine Theory*, v. 38, p. 767–776, 2003.

KAMM, L. J. *Designing Cost-Efficient Mechanisms*. [S.l.]: Mc Graw Hill, 1990.

LEDUC, S. A. *Linear Algebra*. [S.l.]: Wiley Publishing, Inc., 1996.

MARTINS, D. *Análise Cinemática Hierárquica de Robôs Manipuladores*. Tese (Doutorado) — Universidade Federal de Santa Catarina, 2002.

MUROTA, K. *Matrices and Matroids for Systems Analysis*. [S.l.]: Springer, 2000.

POLLARD, A. F. C. Kinematic design in engineering. *Proceedings of the Institution of Mechanical Engineers*, v. 125, p. 143–195, 1933.

POOLE, D. *Linear Algebra A Modern Introduction*. second. [S.l.]: Thomson Brooks/Cole, 2006.

RESHETOV, L. *Self-Aligning Mechanism*. 2. ed. [S.l.]: Mir, 1979.

RICO, J. M.; GALLARDO, J.; RAVANI, B. Lie algebra and the mobility of kinematic chains. *Journal of Robotic Systems*, v. 20, n. 8, p. 477–499, 2003.

SCHELLEKENS, P. et al. Design for precision: Current status and trends. *Annals of the CIRP*, v. 47, n. 2, p. 557–586, 1998.

SHUKLA, G.; WHITNEY, D. E. The path method for analyzing mobility and constraint of mechanism and assemblies. *IEEE Transactions on Automation Science and Engineering*, v. 2, n. 2, p. 184–192, April 2005.

SICILIANO, B. et al. *Robotics Modeling, Planning and Control*. [S.l.]: Springer, 2009.

SIEK, J. G.; LEE, L. Q.; LUMSDAINE, A. *The Boost Graph Library user Guide and Reference Manual*. Bosto, MA, USA: Addison-Wesley Longman Publishing Co., Ing., 2002.

SKAKOON, J. G. *Detailed Mechanical Design: A Practical Guide*. [S.l.]: ASME Press, 2000.

SKAKOON, J. G. Exact constraint. *ME Magazine*, p. 6, September 2009.

SMITH, D. K. *Constraint Analysis of Assemblies Using Screw Theory and Tolerance Sensitivities*. Tese (Master of Science Thesis) — Brigham Young University, Department of Mechanical Engineering, December 2001.

SZYDLOWSKI, W. M. *Self-Aligning Mechanisms, Forgotten Part of ME Curriculum*. [S.l.], 2000.

TSAI, L. W. *The Mechanics of Serial and Parallel Manipulators*. [S.l.]: John Wiley and Sons, Inc., 1999.

TSAI, L. W. *Mechanism Design: enumeration of kinematic structures according to function*. [S.l.]: CRC Press, 2001.

WHITNEY, D. E. *Mechanical Assemblies: Their Design, Manufacture, and Role in Product Development*. [S.l.]: Oxford, 2004.

WHITNEY, D. E. et al. Toward a theory for design of kinematically constrained mechanical assemblies. *The International Journal of Robotics Research*, n. 18, p. 1235–1248, 1999.

YANG, W. Y.; LEE, S. C. *Circuit Systems with Matlab and PSpice*. [S.l.]: John Wiley and Sons, Inc., 2007.

APPENDIX A – Mozzi Axis

In instantaneous kinematics, for each unitary freedom $\boldsymbol{f_i}$ there will be a corresponding motion screw $\$^{\boldsymbol{M}}$. In (BALL, 1998) is explained how Mozzi (1763) describes the infinitesimal displacement of a body as an angular differential velocity $\vec{\boldsymbol{\omega}}$ and a linear velocity $\vec{\boldsymbol{v}}$ both being related to the same screw axis or *Mozzi axis*. The angular velocity is considered to be the line vector defined by the axis of rotation, while the free vector is represented by the linear velocity vector. The twist represented in equation A.1 indicates how the vectors are related, were the pitch ($\boldsymbol{h}$) of the screw is obtained in equation A.2.

$$\$^{\boldsymbol{M}} = \begin{bmatrix} \vec{\boldsymbol{\omega}} \\ \vec{\boldsymbol{v}} \end{bmatrix} \tag{A.1}$$

$$\boldsymbol{h} = \frac{|\vec{\boldsymbol{v}}|}{|\vec{\boldsymbol{\omega}}|} \tag{A.2}$$

The vector $\vec{\boldsymbol{S}}^{\boldsymbol{M}}$ represents the directory cosines in the normalized twist $\hat{\$}^{\boldsymbol{M}}$ is considered to be the axis of the screw, therefore such vector is the line vector. The Plücker coordinates for motion screws are showed in the equation A.3, and in equation A.4 the coordinates for the normalized twist. In both, the coordinates $\boldsymbol{(L, M, N)}$ in instantaneous kinematics represent the line vector and $\boldsymbol{(P^*, Q^*, R^*)}$ the free vector. This formation is called axial formation (MARTINS, 2002). The directory cosines vector related the coordinates by $\boldsymbol{L^2 + M^2 + N^2 = 1}$ and the magnitude of the vector $\boldsymbol{\phi}$, is obtained by: $\boldsymbol{\phi}$=$|\vec{\boldsymbol{\omega}}|$.

$$\$^{\boldsymbol{M}} = \boldsymbol{(L, M, N; P^*, Q^*, R^*)} \tag{A.3}$$

$$\hat{\$}^{\boldsymbol{M}} = \boldsymbol{(\hat{L}, \hat{M}, \hat{N}; \hat{P}^*, \hat{Q}^*, \hat{R}^*)} \tag{A.4}$$

A normalized twist $\hat{\$}^{\boldsymbol{M}}$ is obtained by separating the geometrical twist from its greatness $\boldsymbol{\phi}$, as mentioned before. The equation A.5 separates the unitary twist from its greatness and uses the position and motion vectors to form the twist.

$$
\begin{aligned}
\$^M &= \begin{bmatrix} \boldsymbol{L} \\ \boldsymbol{M} \\ \boldsymbol{N} \\ \boldsymbol{P^* = P + hL} \\ \boldsymbol{Q^* = Q + hM} \\ \boldsymbol{R^* = R + hN} \end{bmatrix} \\
&= \begin{bmatrix} \vec{\omega} \\ \vec{S_0} \times \vec{\omega} + h\vec{\omega} \end{bmatrix} \\
&= \begin{bmatrix} \vec{S}^M \phi \\ (\vec{S_0} \times \vec{S}^M + h\vec{S}^M)\phi \end{bmatrix} \\
&= \begin{bmatrix} \vec{S}^M \\ \vec{S_0} \times \vec{S}^M + h\vec{S}^M \end{bmatrix} \phi \\
&= \hat{\$}^M \phi
\end{aligned}
\tag{A.5}
$$

The pitch $\boldsymbol{h}$ is a variable that indicates in the sign either, positive or negative, the direction of the screw according to the right hand rule (HUNT, 1978). The relation between the angular and linear velocity quantifies the longitudinal and angular displacement of a body. The pitch has two particular situations, when $\boldsymbol{h}$=**0** and when $\boldsymbol{h}$=$\boldsymbol{\infty}$. When the pitch is equal to zero pure rotation, just like an R-pair, is described as the motion of the rigid body. A zero pitch means that the magnitude of $\vec{\boldsymbol{v}}$ is too small compared to the magnitude of $\vec{\boldsymbol{\omega}}$ as showed in equation A.2. A zero pitch twist is showed in equation A.6, and in equation A.7 the magnitude of the velocities is obtained.

$$
\$^M = \begin{bmatrix} \boldsymbol{L} \\ \boldsymbol{M} \\ \boldsymbol{N} \\ \boldsymbol{P} \\ \boldsymbol{Q} \\ \boldsymbol{R} \end{bmatrix} = \begin{bmatrix} \vec{\omega} \\ \vec{S_0} \times \vec{\omega} \end{bmatrix} = \begin{bmatrix} \vec{S}^M \\ \vec{S_0} \times \vec{S}^M \end{bmatrix} \phi \tag{A.6}
$$

$$
\phi = |\vec{\omega}| = \sqrt{\boldsymbol{L^2 + M^2 + N^2}} \tag{A.7}
$$

On the other side, the pitch can be so big that it is considered equal to infinite. The motion of the rigid body can be compared to a linear translation, just like a P-pair. In equation A.2 the magnitude of $\vec{\boldsymbol{\omega}}$ is to small compared with the magnitude of $\vec{\boldsymbol{v}}$ resulting in the linear translation. The equation of the screw with infinite pitch is in equation

A.8 and in equation A.9 the magnitude of $\$^{M}$ is obtained.

$$\$^{M} = \begin{bmatrix} \mathbf{0} \\ \mathbf{0} \\ \mathbf{0} \\ \boldsymbol{P} \\ \boldsymbol{Q} \\ \boldsymbol{R} \end{bmatrix} = \begin{bmatrix} \mathbf{0} \\ \vec{v} \end{bmatrix} = \begin{bmatrix} \mathbf{0} \\ \vec{S}^{M} \end{bmatrix} \phi \tag{A.8}$$

$$\phi = |\vec{v}| = \sqrt{\boldsymbol{P^2} + \boldsymbol{Q^2} + \boldsymbol{R^2}} \tag{A.9}$$

APPENDIX B – Poinsot Axis

In (BALL, 1998) explains Poinsot (1806) proved that for any rigid body with different applied forces and couples, there is always an axis where a resultant force vector $\vec{R}_R$ [$distance$] and a resultant moment $\vec{T}_R$ [$force \times distance$] act in the body without changing its static equilibrium (DAVIES, 2000). This axis, known as *Poinsot axis*, might not even be on the body, but it will match with the action screw axis, or wrench $\$^A$. The wrench is formed by a line vector indicating the resultant force $\vec{R}_R$ and by a free vector representing the resultant moment $\vec{T}_R$. The moment $\vec{T}_R$, follows the right hand rule and it is represented by a binary force. This is, a couple of forces with the same magnitude as $\vec{R}_R$, that act on a perpendicular plane to $\vec{T}_R$. The two forces are separated by a distance d and they are directed parallel to the Poinsot axis, see illustration a) in Figure 25. The binary force has a magnitude and is free of orientation on the plane therefore they can be classified as the free vector on the action screw. The wrench is formed by a line vector $\vec{R}_R$ and a free vector $\vec{T}_R$ which are represented in the action screw as shows equation B.1, and in equation B.2 the magnitude of the vectors are divided to know the value of the pitch.

$$\$^A = \begin{bmatrix} \vec{R}_R \\ \vec{T}_R \end{bmatrix} \tag{B.1}$$

$$h = \frac{|\vec{T}_R|}{|\vec{R}_R|} \tag{B.2}$$

The Plücker coordinates for action screws are showed in equation B.3, such formation is called radial formation (MARTINS, 2002). A normalized wrench $\hat{\$}^A$ can also be obtained by separating the directory cosines from any greatness ψ. In equation B.4 the coordinates for a unitary action screw. In statics the coordinates (P, Q, R) corresponds to the binary moment vector, it is the free vector and the coordinates (L, M, N) to the line vector. The directory cosines coordinates, represented by $\vec{S}^A$, are related by $L^2 + M^2 + N^2 = 1$, and the magnitude of the wench is the vector ψ=$|\vec{R}_R|$.

$$\$^A = (P, Q, R; L^*, M^*, N^*) \tag{B.3}$$

$$\hat{\$}^A = (\hat{P}, \hat{Q}, \hat{R}; \hat{L}^*, \hat{M}^*, \hat{N}^*) \tag{B.4}$$

A normalized wrench $\hat{\$}^A$ is obtained by separating it from its

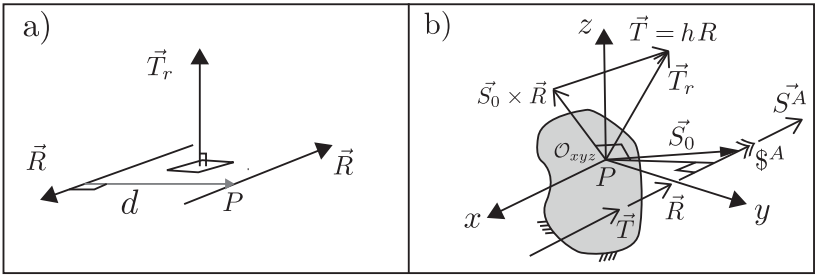


Figure 25 – Wrench, geometric representation, taken from (CAZANGI, 2008) Fig 3.8 and 3.9

greatness $\boldsymbol{\psi}$. The obtention of the action screw and its magnitude is done as showed in equation B.5.

$$
\begin{aligned}
\$^{\boldsymbol{A}} &= \begin{bmatrix} \boldsymbol{P^* = P + hL} \\ \boldsymbol{Q^* = Q + hM} \\ \boldsymbol{R^* = R + hN} \\ \boldsymbol{L} \\ \boldsymbol{M} \\ \boldsymbol{N} \end{bmatrix} \\
&= \begin{bmatrix} \vec{\boldsymbol{S}}_{\boldsymbol{0}} \times \vec{\boldsymbol{R}}_{\boldsymbol{R}} + \boldsymbol{h}\vec{\boldsymbol{R}}_{\boldsymbol{R}} \\ \vec{\boldsymbol{R}}_{\boldsymbol{R}} \end{bmatrix} \\
&= \begin{bmatrix} (\vec{\boldsymbol{S}}_{\boldsymbol{0}} \times \vec{\boldsymbol{S}}^{\boldsymbol{A}} + \boldsymbol{h}\vec{\boldsymbol{S}}^{\boldsymbol{A}})\boldsymbol{\psi} \\ \vec{\boldsymbol{S}}^{\boldsymbol{A}}\boldsymbol{\psi} \end{bmatrix} \\
&= \begin{bmatrix} \vec{\boldsymbol{S}}_{\boldsymbol{0}} \times \vec{\boldsymbol{S}}^{\boldsymbol{A}} + \boldsymbol{h}\vec{\boldsymbol{S}}^{\boldsymbol{A}} \\ \vec{\boldsymbol{S}}^{\boldsymbol{A}} \end{bmatrix} \boldsymbol{\psi} \\
&= \hat{\$}^{\boldsymbol{A}}\boldsymbol{\psi}
\end{aligned}
\tag{B.5}
$$

In the wrench the relation between the force and the moment is given by the pitch $\boldsymbol{h}$, it presents two particular situation, ether is it zero $\boldsymbol{h}$=**0** or it is infinite $\boldsymbol{h}$=$\boldsymbol{\infty}$. If the wrench has pitch zero $\boldsymbol{h} = \mathbf{0}$, then pure force is acting in straight line on a body, and it is showed in equation B.6. The result of such zero comes from the difference between $\vec{\boldsymbol{T}}_{\boldsymbol{R}}$ and $\vec{\boldsymbol{R}}_{\boldsymbol{R}}$, demonstrated in showed in equation A.2, in equation B.7 the magnitude of the couple is too small compared to the vector force.

$$\$^{A} = \begin{bmatrix} P \\ Q \\ R \\ L \\ M \\ N \end{bmatrix} = \begin{bmatrix} \vec{S_0} \times \vec{R}_R \\ \vec{R}_R \end{bmatrix} = \begin{bmatrix} \vec{S_0} \times \vec{S}^A \\ \vec{S}^A \end{bmatrix} \psi \tag{B.6}$$

$$\psi = |\vec{R}_R| = \sqrt{L^2 + M^2 + N^2} \tag{B.7}$$

A wrench with infinite pitch, $h = \infty$, represents a pure couple acting on a body, it is showed in equation B.8 and equation B.8 to determine the magnitude of the action screw.

$$\$^{A} = \begin{bmatrix} P^* \\ Q^* \\ R^* \\ 0 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} \vec{T}_R \\ \vec{0} \end{bmatrix} = \begin{bmatrix} \vec{S}^A \\ \vec{0} \end{bmatrix} \psi \tag{B.8}$$

$$\psi = |\vec{T}_R| = \sqrt{L^2 + M^2 + N^2} \tag{B.9}$$

The kinematic chains, graphs and screws will be used to describe a mechanisms, and they will set the parameters to perform a linear independence analysis. In chapter 4 some of the presented methodologies use these concepts to determine the state of constraint of the mechanical system.

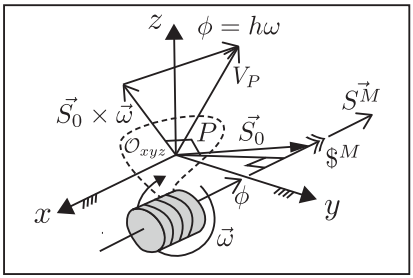


Figure 26 – Twist geometric representation, taken by (CAMPOS, 2004) Fig. 10.

APPENDIX C – Dulmage and Mendelsohn Decomposition

In Graph Theory a bipartite graph is when the vertices of a graph can be divided into two independent sets of vertices, for example $\boldsymbol{V}^+$ and $\boldsymbol{V}^-$. In this way edges go from one set to the other, there are no edges connecting vertices from the same subset. Bipartite graphs can be used to relate a set of variables with a set of equations (BUNUS; FRITZSON, 2002) for example. In Figure 27 a bipartite graph is illustrated, notice that not all the vertices from one set connect with the vertices of the other set. If all the vertices from $\boldsymbol{V}^+$ match with all the vertices from $\boldsymbol{V}^-$, then it would be a bipartite graph with perfect matching.

This research use the Dulmage-Mendelsohn (DM) Decomposition, which is implemented in GNU Octave to obtain a block upper triangular form of the motion network and action network matrices. Such decomposition is used since it returns identifiable blocks that will be further analyze for linear dependency. This is possible to realize since the DM decomposition finds a maximum matching in a bipartite graph (MUROTA, 2000).

To understand the DM Decomposition consider the matrix $\mathbf{A}$ shown in C.1. The bipartite graph of such matrix is shown in figure 27. The graph is defined as $\boldsymbol{G} = \{\boldsymbol{V}^+, \boldsymbol{V}^-; \boldsymbol{E}\}$. This means that the graph $\boldsymbol{G}$ is formed by $\boldsymbol{V}^+$ that is a set of vertices that represent the columns the matrix, $\boldsymbol{V}^-$ is also a set representing the lines of the same matrix, and a set of $\boldsymbol{E}$ edge that connect the sets of vertices based on the relationship between the lines and the columns observed in the matrix.

$$\mathbf{A} = \begin{bmatrix} \mathbf{0} & \mathbf{0} & \mathbf{5} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{3} & \mathbf{4} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{5} & \mathbf{2} \\ \mathbf{1} & \mathbf{0} & \mathbf{0} & \mathbf{2} & \mathbf{0} & \mathbf{0} & \mathbf{3} & \mathbf{0} & \mathbf{0} & \mathbf{8} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{6} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{9} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{7} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{1} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{8} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{4} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \end{bmatrix} \tag{C.1}$$

The DM Decomposition will return a family of subgraphs $\boldsymbol{G_k} = \{\boldsymbol{V_k^+}, \boldsymbol{V_k^-}; \boldsymbol{E_k}\}$ were $\boldsymbol{k}$ can take any value between $\mathbf{0}$ and $\boldsymbol{\infty}$ as described in (MUROTA, 2000). The maximum matching bipartite graph for this example is shown in 28.

The matrix representation of the maximum matching graph, for this example is shown in C.2. The decomposition returns the finest block-triangular, using row and column permutations (DULMAGE; MENDELSOHN, 1958). DM Decomposition implemented in GNU Octave

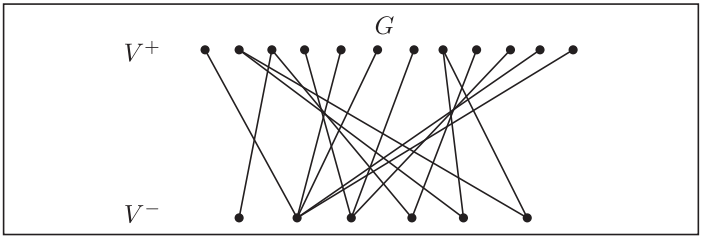


Figure 27 – Bipartite graph of the matrix **A** shown in matrix C.1

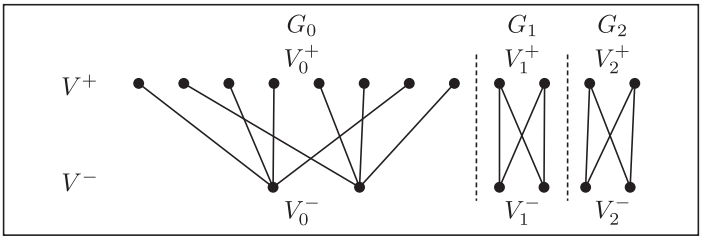


Figure 28 – Bipartite graph of maximum matching **A** shown in matrix C.1

returns from a given matrix, four line vectors. The first and second vector de indicate the line and column permutation respectively, the third and fourth vector indicate were are the limits between the blocks in the maximum matching matrix. Each block of the triangular has its own rank, defined as term-rank. Term-rank coincide with the rank of a generic matrix, if all nonzero entires are independent parameters (MUROTA, 2000), it also explained that even when the term-rank is a natural combinatorial counterpart and it is not the same as the rank of the matrix.

$$\mathbf{A} = \begin{array}{c|cccccccc|cc|cc} & \multicolumn{8}{c|}{\boldsymbol{v_0^+}} & \multicolumn{2}{c|}{\boldsymbol{v_1^+}} & \multicolumn{2}{c}{\boldsymbol{v_2^+}} \\ \boldsymbol{v_0^-} & \mathbf{2} & \mathbf{0} & \mathbf{3} & \mathbf{8} & \mathbf{0} & \mathbf{0} & \mathbf{1} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ & \mathbf{0} & \mathbf{4} & \mathbf{0} & \mathbf{0} & \mathbf{5} & \mathbf{2} & \mathbf{0} & \mathbf{3} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \hline \boldsymbol{v_1^-} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{7} & \mathbf{1} & \mathbf{0} & \mathbf{0} \\ & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{8} & \mathbf{4} & \mathbf{0} & \mathbf{0} \\ \hline \boldsymbol{v_2^+} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{5} & \mathbf{8} \\ & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{6} & \mathbf{9} \end{array} \tag{C.2}$$

APPENDIX D – Action matrices for Case I and Case II.

This appendix show the action matrices for both cases mentioned in section 4.2 were the algorithm is described.

$$
[A_N] = \begin{array}{c}
\\
P_1 \\ Q_1 \\ R_1 \\ L_1 \\ M_1 \\ N_1 \\ P_2 \\ Q_2 \\ R_2 \\ L_2 \\ M_2 \\ N_2 \\ P_3 \\ Q_3 \\ R_3 \\ L_3 \\ M_3 \\ N_3
\end{array}
\begin{array}{c}
\begin{array}{cccccccccccccccccccc}
a1 & a2 & a3 & a4 & a5 & b1 & b2 & b3 & b4 & b5 & c1 & c2 & c3 & c4 & c5 & d1 & d2 & d3 & d4 & d5
\end{array} \\
\left[\begin{array}{cccccccccccccccccccc}
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 6 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & -6 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 6 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 2 & -2 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & -6 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 6 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & -6 & 0 & 1 & 0 & -6 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1
\end{array}\right]_{(18,20)}
\end{array}
\tag{D.1}
$$

$$
[\boldsymbol{A_N}]_{m\text{x}m} =
\begin{array}{c|cccccccccccccccc|c|c|c|c}
 & d1 & d2 & d3 & d4 & a2 & a3 & a4 & a5 & b2 & b3 & b4 & b5 & c2 & c3 & c4 & c5 & a1 & b1 & c1 & d5 \\
\hline
M_1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 6 \\
N_1 & 0 & 1 & 0 & -6 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
P_1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
Q_1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
M_2 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 6 \\
N_2 & 0 & 1 & 0 & -6 & 0 & 0 & 0 & 0 & 0 & 1 & 2 & -2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
P_2 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
Q_2 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
M_3 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 6 \\
N_3 & 0 & 1 & 0 & -6 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & -6 & 0 & 0 & 0 & 0 \\
P_3 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
Q_3 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
\hline
L_1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
\hline
L_2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
\hline
L_3 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
\hline
R_1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
R_2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
R_3 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1
\end{array}_{(18,20)}
\tag{D.2}
$$

$$Dep = \left[\begin{array}{c|c|c|c|c} \{d1\}\{d2\}\{d3\}\{d4\}\{a2\}\{a3\}\{a4\}\{a5\}\{b2\}\{b3\} & & & & \\ \{b4\}\{b5\}\{c2\}\{c3\}\{c4\}\{c5\}\{d1,\ d2\}\{d1,\ d3\}\{d1,\ d4\}\{d1,\ a3\} & & & & \\ \{d1, a4\}\{d1, a5\}\{d1, b3\}\{d1, b4\}\{d1, b5\}\{d1, c3\}\{d1, c4\}\{d1, c5\}\{d2, d3\}\{d2, d4\} & & & & \\ \{d2, a2\}\{d2, a3\}\{d2, a4\}\{d2, a5\}\{d2, b2\}\{d2, b3\}\{d2, b4\}\{d2, b5\}\{d2, c2\}\{d2, c4\} & & & & \\ \{d2, c5\}\{d3, d4\}\{d3, a2\}\{d3, a3\}\{d3, a5\}\{d3, b2\}\{d3, b3\}\{d3, b5\}\{d3, c2\}\{d3, c3\} & & & & \\ \{d3, c5\}\{d4, a2\}\{d4, a3\}\{d4, a4\}\{d4, b2\}\{d4, b3\}\{d4, b4\}\{d4, c2\}\{d4, c3\}\{d4, c4\} & & & & \\ \{a2, a3\}\{a2, a4\}\{a2, a5\}\{a2, b3\}\{a2, b4\}\{a2, b5\}\{a2, c3\}\{a2, c4\}\{a2, c5\}\{a3, a4\} & & & & \\ \{a3, a5\}\{a3, b2\}\{a3, b3\}\{a3, b4\}\{a3, b5\}\{a3, c2\}\{a3, c4\}\{a3, c5\}\{a4, a5\}\{a4, a5\} & & & & \\ \{a5, c3\}\{a5, c4\}\{b2, b3\}\{b2, b4\}\{b2, b5\}\{b2, c3\}\{b2, c4\}\{b2, c5\}\{b3, b4\}\{b3, b5\} & & & & \\ \{b3, c2\}\{, b3c3\}\{b3, c4\}\{b3, c5\}\{b4, b5\}\{b4, c2\}\{b4, c3\}\{b4, c5\}\{b5, c2\}\{b5, c3\} & & & & \\ \{b5, c4\}\{c2, c3\}\{c2, c4\}\{c2, c5\}\{c3, c4\}\{c3, c5\}\{c4, c5\} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \hline \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \hline \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \hline \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \hline \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \end{array}\right] \tag{D.3}$$

$$Indep = \left[\begin{array}{c|c|c|c|c} \{d1, a2\}\{d1, b2\}\{d1, c2\}\{d2, c3\}\{d3, a4\}\{d3, b4\}\{d3, c4\}\{d4, a5\}\{d4, b5\}\{d4, c5\} & & & & \\ \{a2, b2\}\{a2, c2\}\{a4, b4\}\{a4, c4\}\{a5, b5\}\{a5, c5\}\{b2, c2\}\{b4, c4\}\{b5, c5\} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \{d5\} \\ \hline \mathbf{0} & \{a1\} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \hline \mathbf{0} & \mathbf{0} & \{b1\} & \mathbf{0} & \mathbf{0} \\ \hline \mathbf{0} & \mathbf{0} & \mathbf{0} & \{c1\} & \mathbf{0} \\ \hline \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \{d5\} \end{array}\right] \tag{D.4}$$

$$
[\boldsymbol{A_N}] = \begin{array}{c} \\ P_1 \\ Q_1 \\ R_1 \\ L_1 \\ M_1 \\ N_1 \\ P_2 \\ Q_2 \\ R_2 \\ L_2 \\ M_2 \\ N_2 \\ P_3 \\ Q_3 \\ R_3 \\ L_3 \\ M_3 \\ N_3 \end{array}
\begin{array}{c}
\begin{array}{ccccccccccccccccc}
a1 & a2 & a3 & a4 & a5 & b1 & b2 & b3 & c1 & c2 & c3 & c4 & c5 & d1 & d2 & d3 & d4
\end{array} \\
\left[\begin{array}{ccccccccccccccccc}
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 6 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & -6 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 6 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & -6 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 6 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & -6 & 0 & 1 & -6 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1
\end{array}\right]_{\mathbf{(18,17)}}
\end{array}
\tag{D.5}
$$

$$[A_N]_{m\times m} =$$

	$d1$	$d2$	$a2$	$a3$	$b2$	$b3$	$c2$	$c3$	$a1$	$a4$	$a5$	$b1$	$c1$	$c4$	$c5$	$d3$	$d4$
M_1	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	6
N_1	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	−6	0
M_2	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	6
N_2	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	−6	0
M_3	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
N_3	0	1	0	0	0	0	0	1	0	0	0	0	0	0	−6	−6	0
L_1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
P_1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
Q_1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	1	0
L_2	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
L_3	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
P_3	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
Q_3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0
Q_2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
R_1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
P_2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
R_2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
R_3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

(18,17)

(D.6)

$$Dep = \left[\begin{array}{c|ccccccccc} \{d1\}\{d2\}\{a2\}\{a3\}\{b2\} \\ \{b3\}\{c2\}\{c3\}\{d1,d2\}\{d1,d3\} \\ \{d1,b3\}\{d1,c3\}\{d2,a2\}\{d2,b2\}\{d2,c2\} \\ \{a2,a3\}\{a2,b3\}\{a2,c3\}\{a3,b2\}\{a3,c2\} \\ \{b2,b3\}\{b2,c3\}\{b3,c2\}\{c2,c3\} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{array}\right] \tag{D.7}$$

$$Indep = \left[\begin{array}{c|ccccccccc} \{d1,a2\}\{d1,b2\}\{d1,c2\}\{d2,a3\}\{d2,b3\} \\ \{d2,c3\}\{a2,b2\}\{a2,c2\}\{a3,b3\}\{a2,c3\} \\ \{b2,c2\}\{b3,c3\} & 0 & 0 & 0 & 0 & 0 & 0 & \{c5\} & \{d3\} & \{d4\} \\ \hline 0 & \{a1\} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \hline 0 & 0 & \{a4\} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & \{a5\} & 0 & 0 & 0 & 0 & \{d3\} & 0 \\ \hline 0 & 0 & 0 & 0 & \{b1\} & 0 & 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 & 0 & \{c1\} & 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 & 0 & 0 & \{c4\} & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 & 0 & 0 & 0 & \{c5\} & \{d3\} & 0 \\ \hline 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \{d3\} & 0 \\ \hline 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \{d4\} \end{array}\right] \tag{D.8}$$

APPENDIX E – Twist - $_m.m

```
function [ MD,plucker,id_$_m] = $_m ( h_m,Sm,S0_m,BM)
# Written by: Victor Carreto - 2010
# This function  creates the twist matrix from the position
# vectors.
# Inputs:
# h_m  (1,F) = the pitch is a binary vector: 1 if it is
# infinite and 0 if not.
# Sm   (3,F) = each column has the direction of the
# screw axis.
# S0_m (3,F) = each column has the position vector
# BM (loops,F) = circuit matrix
#
# The result is a twist matrix MD, the spatiality lambda
# which is fixed to be six so, all the plucker coordinates
# are involved.
# the plucker coordinates.
matrix;
[ l_h, c_h ] = size (h_m);
[l_bm, c_bm] = size (BM);
# THIS CODE is for MD
MD = zeros (6,1);
for i = 1 : c_h
 Sm_i = Sm   (:,i);
 h_i  = h_m  (:,i);
 S0_i = S0_m (:,i);
 if h_i == 0
  pc = cross (Sm_i, S0_i);
  $ = [ Sm_i; pc ]; #  pitch (h = 0)
 else
  $ = [ 0; 0; 0; Sm_i ]; # pitch (h = 1(inf))
 endif
 MD = [ MD $ ];
endfor
MD (:,1) = [ ];
# THIS CODE is for plucker
aux = l_bm*6;
plucker = id_plucker_m(1:aux,1);
# THIS CODE is for id
id_$_m = id_$ (1,1:c_h);
endfunction
```

APPENDIX F – Wrench - $_a.m

```
function [AD,plucker_a,id_$_a] = $_a ( h_a,Sa,S0_a,QA)
# Written by: Victor Carreto - 2010
# This function  creates the twist matrix from the position
# vectors.
# Inputs:
# h_a  (1,F) = the pitch is a binary vector: 1 if it is
# infinite and 0 if not.
# Sa   (3,F) = each column has the direction of the
# screw axis.
# S0_a (3,F) = each column has the position vector
# BM (loops,F) = circuit matrix
#
# The result is a twist matrix MD, the spatiality lambda
# which is fixed to be six so, all the plucker coordinates
# are involved.
# the plucker coordinates.
matrix;
[ l_h, c_h ] = size (h_a);
[l_qa, c_qa] = size (QA);
# THIS CODE is for AD
AD = zeros ( 6, 1 );
for i = 1 : c_h
 Sa_i = Sa   (:,i);
 h_i  = h_a  (:,i);
 S0_i = S0_a (:,i);
 if h_i == 0
  pc = cross (Sa_i, S0_i);
  $ = [ pc ; Sa_i ];     #  pitch (h = 0)
 else
  $ = [ Sa_i; 0; 0; 0 ]; # pitch (h = 1(inf))
 endif
  AD = [ AD $ ];
endfor
AD (:,1) = [ ];
# THIS CODE is for plucker
aux = l_qa*6;
plucker_a = id_plucker_a(1:aux,1);
# THIS CODE is for id
id_$_a = id_$ (1,1:c_h);
endfunction
```

APPENDIX G – davies.m

```
function [N,rank_N] = davies (D, M)
# Written by: Victor Carreto - 2010
# This functions composes a Network matrix according to the
# Davies Method.
# Inputs = D (MD or AD), M (BM or QA)
# D (lambda, num_$) = screw matrix of the direct couplings
# M (lines, num_$)  = circuit matrix or cut-set matrix
# Output
# N (lambda*lines,num_$) = network  matrix.
# rank_N = number of independent vectors.
# variable inicialization
v  = [0];
n  = [0];
a  = [0];
N0 = [0];
# to know how many rows and columns the inputs have
[num_circuitos,num_pares] = size (M);
[lambda,num_heligiros] = size (D);
# the main idea is to take each and every row of 'M' and
# diagonalize the vector forming a 'mat_diag' matrix wich
# will multiply 'D'. Then we will put each partial
# result under the previos one.
for i = 1:num_circuitos;
v = M(i,:);
mat_diag = diagonal (v);
n = D * mat_diag;
if N0 == [0];
N0 = n;
else
N0 = [N0;n];
endif
endfor
# printing the result
N = N0;
rank_N = rank(N);
endfunction
```

APPENDIX H – state.m

```
function state (FN,CN)
# Written by: Victor Carreto - 2010
# This function returns a leyend indicating the state of
# constraint of a mechanism.
# inputs:
# FN = is a constant indicating the nett Freedoms
# CN = is a constant indicating the nett Constrains.
if (CN == 0) & (FN > 0)
 printf("\n")
 printf("++++++++++++++++++++++++++++++++++++++++\n")
 printf("State: \n")
 printf("KINEMATICALLY DESIGNED Kinematic Chains\n")
 printf("++++++++++++++++++++++++++++++++++++++++\n")
elseif (CN == 0) & (FN == 0)
 printf("\n")
 printf("+++++++++++++++\n")
 printf("State: \n")
 printf("RIGID structure\n")
 printf("+++++++++++++++\n")
elseif (CN > 0) & (FN == 0)
 printf("\n")
 printf("++++++++++++++++++++++++++\n")
 printf("State: \n")
 printf("OVERCONSTRAINED structure\n")
 printf("++++++++++++++++++++++++++\n")
elseif (CN > 0) & (FN > 0)
 printf("\n")
 printf("++++++++++++++++++++++++++++++++\n")
 printf("State: \n")
 printf("OVERCONSTRAINED Kinematic Chain\n")
 printf("++++++++++++++++++++++++++++++++\n")
else
  state = 0;
endif
endfunction
```

APPENDIX I – kinematic.m

```
function kinematic_analysis (h_m, Sm, S0_m, BM )
# Written by: Victor Carreto - 2010
# kinematic_analysis (h_m, Sm, S0_m, BM )
# This function performs an instantaneous kinematic analysis
# of a given mechanis.
# Inputs:
#  h_m (1,F)   = binary vector: 1 if the pitch is infinite
# and 0 if not.
#  Sm (3,F)    = matrix were each column indicate the
# direction of the screw axis.
#  S0_m (3,F)  = matrix were each column indicate the
# position vector.
# BM (loops,F) = circuit matrix.
# Outputs
#  indep = array indicating in each cell a set of independent
# vectors
#  id_indep = array indicating in each cell a set of indep.
# identification letters indicating each freedom or constraint
#  dep = array indicating in each cell a set of dependent
# vectors
#  id_dep = array indicating in each cell a set of dep.
# identification letters indicating each freedom or constraint
# The function heligiros.m is used to create the matrix MD
# and indicates the value of lambda which is pretermined to
# be six, and the plucker coordinates.
[MD,plucker_MN,id_$_m] = $_m (h_m,Sm,S0_m,BM);
# The function MN_v1.m combines the MD, and BM to obtain
# the Motion Network matrix and its rank.
[MN,rank_MN] = davies (MD, BM);
# Step 1 - Print:
# Screw Matrix MD,
# Motion Network MN,
# Rank of MN,
# Number of loops,
# Gross degree of freedom,
# Nett constraints
# Nett freedoms. it represents the number of
# PRIMARY VARIABLES.
[l_bm,c_bm]=size(BM);
printf('\n')
# printf('PRINT 1\n')
MD;
MN
rank_MN
plucker_MN
```

```
id_$_m
loops = l_bm
F = c_bm
CN = 6*loops - rank_MN
FN = F-rank_MN
# Funtion state.m require the nett freedom and the nett
# constraint of the network, it return the state of
# constraint of the mechanism. It can be: overconstraint
# structure or overconstraint kinematic chain or
# kinematically disigned
# kinematic chain or rigid structure).
printf('\n')
state (FN,CN)
printf('\n')
printf('---------------------------------------- \n')
printf('    Searching for maximum matching...    \n')
printf('---------------------------------------- \n')
# Function dmperm.m requires a matrix and it will perform
# the Dulmage-Mendelsohn permutation, and it will return
# four line vectors the fist two indicate the row and column
# permutation to obtain a block triangular of the given
# matrix, the second two define the limits of such blocks
[p,q,r,s] = dmperm(MN);
# Step 2 - Motion network matrix of maximum matching,
# the rank and the vectors that to keep identified the
# lines and the columns.
# plucker_mxm and twist_mxm are identification vectors.
printf('\n')
printf('PRINT 2\n')
printf('\n')
MN_mxm = MN (p,q)
rank_MN_mxm = rank (MN_mxm)
plucker_mxm = plucker_MN (p,:)
twist_mxm = id_$_m(:,q)
if FN == 0
 printf('all columns are independent\n')
else
 printf('number of dependencies in MN_mxm\n')
 pv = FN
# The following code identifies the amount of bolocks and
# the returned variables are arrays were each cell contains
# the information of the identified blocks,
[blocks,id_lin,id_col]=id_blocks(MN_mxm,p,q,r,s,plucker_mxm,
 twist_mxm);
 [l_block,c_block]=size(blocks);
```

```
# This code separates each block and its identification
# vectors from the array that contains them, it also obtains
# the number of primary variables 'pv' of the block.
# primary variables = number of dependent columns
 for i=1:c_block
 printf('\n')
 printf('========================================= \n')
 printf('+++++++++++++++++++++++++++++++++++++++++ \n')
 printf('========================================= \n')
 identified_block_i = i
 printf('========================================= \n')
 printf('+++++++++++++++++++++++++++++++++++++++++ \n')
 printf('========================================= \n')
 bi=blocks{1,i}
 id_lin_i=id_lin{1,i}
 id_col_i=id_col{1,i}
 t_rank = rank(bi)
 n_c_i = columns(bi);
 pv_i = n_c_i - t_rank
 # if the primary variables are zero it means that all the
 # columns in the block are independent from each other.
 if pv_i==0
  printf('\n')
  printf('+++++++++++++++++++++++++++++++++ \n')
  printf('  All Independent columns :  \n')
  printf('+++++++++++++++++++++++++++++++++ \n')
  c_1 = bi;
  id_1 = id_col_i;
  c_0 = [ ];
  id_0 = [ ];
  c_indep = bi;
####################
  dep{1,i} = [ ];
  id_dep{1,i} = [ ];
  indep{1,i} = bi;
  id_indep{1,i} = id_col_i;
####################
 else
# the block has one or more primary variables, this code
# calls a function to find columns of zeros that justify
# the number of primary variables.
# the founded columns of zero are considered dependent and
# those non zero columns are considered independent.
  clear [c_0 id_0 nc_0 c_1 id_1 nc_1];
bi;
```

```
  [c_0,id_0,nc_0,c_1,id_1,nc_1]=find_col_zeros(bi,id_col_i);
 nc_0
  if nc_0 == pv_i
   printf('\n')
   printf('******************************** \n');
   printf(' ALL Dependent columns are zero: \n');
   printf('******************************** \n');
   c_dep = c_0;
   nc_dep = columns(c_1);
   id_dep_0 = id_0;
   printf('++++++++++++++++++++++++++++++++ \n');
   printf('  ALL Independent columns found:  \n');
   printf('++++++++++++++++++++++++++++++++ \n');
   c_indep = c_1;
   nc_indep = columns (c_1);
####################
  dep{1,i} = c_0;
  id_dep{1,i} = id_0;
  indep{1,i} = c1;
  id_indep{1,i} = id_1;
####################
  elseif nc_0 == 0
   printf('\n')
   printf('------------------------ \n');
   printf(' NO zero columns found:  \n');
   printf('------------------------ \n');
   c_0 = [ ];
   id_0 = [ ];
   nc_0=0;
   pv_i_r = pv_i;
#   printf('----------------------------------\n');
#   printf(' columns are non-all zero matrix:  \n');
#   printf('----------------------------------\n');
   nc_1;
   c_1;
   id_1;
[c_1_indep,id_1_indep,c_1_dep,id_1_dep]=enc(c_1,id_1,pv_i_r);
   printf('------------------ \n');
   printf('Dependent sets:    \n');
c_1_dep;
id_1_dep;
num_dep = columns (c_1_dep)
   printf('------------------ \n');
   printf(' Independent sets: \n');
   c_1_indep;
```

```
   id_1_indep;
   num_indep = columns (c_1_indep)
####################
  dep{1,i} = c_1_dep;
  id_dep{1,i} = id_1_dep;
  indep{1,i} = c_1_indep;
  id_indep{1,i} = id_1_indep;
####################
# the number of columns of zero did not justify all the
# dependencies in the block, the remaining primary variables
  else
   printf('\n')
   printf('------------------------------\n');
   printf(' dependent zero columns found:  \n');
   printf('------------------------------\n');
   c_dep {1,1} = c_0;
   nc_dep = columns(c_0);
   id_dep {1,1} = id_0;
   nc_0;
   pv_i_r = pv_i-nc_0;
   printf('------------------------------\n');
   printf('remaining non-all zero matrix:\n');
   printf('------------------------------\n');
   nc_1;
   c_1;
   id_1;
[c_1_indep,id_1_indep,c_1_dep,id_1_dep]=enc(c_1,id_1,pv_i_r);
   printf('******************************** \n');
   printf(' Dependencies found in matrix \n');
   printf('******************************** \n');
c_dep {1,2} = c_1_dep;
c_dep {1,2} = id_1_dep;
   printf('++++++++++++++++++++++++++++++++ \n');
   printf(' Independent found in matrix \n');
   printf('++++++++++++++++++++++++++++++++ \n');
   c_1_indep;
   id_1_indep;
####################
  dep{1,i} = c_dep;
  id_dep{1,i} = id_dep;
  indep{1,i} = c_1_indep;
  id_indep{1,i} = id_1_indep;
####################
  endif
 endif
```

```
# indep {1,i} = c_1;
# id_indep {1,i} = id_1;
# dep {1,i} = c_0;
# id_dep {1,i} = id_0;
 endfor
   printf('\n')
   printf('=_=_=_=_=_=_=_=_=_=_=_=_=_=_=_=_=_=_=_=_=_=_= \n');
   printf('\n')
   printf('Set of Independent columns in each block :\n');
   printf('\n')
   printf('=_=_=_=_=_=_=_=_=_=_=_=_=_=_=_=_=_=_=_=_=_=_= \n');
 indep;
 id_indep
   printf('\n')
   printf('=_=_=_=_=_=_=_=_=_=_=_=_=_=_=_=_=_=_=_=_=_=_= \n');
   printf('\n')
   printf('Set of dependent columns in each block :\n');
   printf('\n')
   printf('=_=_=_=_=_=_=_=_=_=_=_=_=_=_=_=_=_=_=_=_=_=_= \n');
 dep;
 id_dep
endif
endfunction
```

APPENDIX J – static.m

```
function static_analysis (h_a, Sa, S0_a, QA )
# Written by: Victor Carreto - 2010
# This function performs an instantaneous static analysis
# of a given mechanis.
#
# Inputs:
#  h_a (1,C)   = binary vector: 1 if the pitch is infinite
# and 0 if not.
#  Sa (3,C)    = matrix were each column indicate the
# direction of the screw axis.
#  S0_a (3,C)  = matrix were each column indicate the
# position vector.
# QA (cuts,C) = circuit matrix.
#
# The result is a serch for dependencies in the
# motion network.
# The function heligiros.m is used to create the matrix MD
# and indicates the value of lambda which is pretermined to
# be six, and the plucker coordinates.
[AD,plucker_AN,id_$_a] = $_a (h_a,Sa,S0_a,QA);
# The function davies.m combines the AD, and QA to obtain
# the Action Network matrix and its .
[AN,rank_AN] = davies (AD, QA);
# Step 1 - print:
# Screw Matrix AD,
# Motion Network AN,
# Rank of AN,
# Number of cut-sets,
# Gross degree of constraint,
# Nett constraints
# Nett freedoms.
[l_qa,c_qa]=size(QA);
printf('\n')
#printf('PRINT 1\n')
AD
AN
rank_AN
plucker_AN
id_$_a
cut_sets = l_qa
C = c_qa
CN = C-rank_AN
FN = 6*cut_sets - rank_AN
# Funtion state.m require the nett freedom and the nett
# constraint of the network, it return the state of
```

```
# constraint of the mechanism. It can be: overconstraint
# structure or overconstraint kinematic chain or
# kinematically disigned
# kinematic chain or rigid structure).
printf('\n')
state (FN,CN)
printf('\n')
printf('---------------------------------------- \n')
printf('    Searching for maximum matching...    \n')
printf('---------------------------------------- \n')
# Function dmperm.m requires a matrix and it will perform
# the Dulmage-Mendelsohn permutation, and it will return
# four line vectors the fist two indicate the row and column
# permutation to obtain a block triangular of the given
# matrix, the second two define the limits of such blocks
[p,q,r,s] = dmperm(AN);
# Step 2 - Action network matrix of maximum matching,
# the rank and the vectors that to keep identified the
# lines and the columns.
# Step 2 - Motion network matrix of maximum matching,
# the rank and the vectors that to keep identified the
# lines and the columns.
# plucker_mxm and wrench_mxm are identification vectors.
printf('\n')
#printf('PRINT 2\n')
AN_mxm = AN (p,q)
rank_AN_mxm = rank (AN_mxm)
plucker_mxm = plucker_AN (p,:)
wrench_mxm = id_$_a(:,q)
if CN == 0
 printf('all columns are independent\n')
else
 printf('number of dependencies in MN_mxm\n')
 pv = CN
# The following code identifies the amount of bolocks and
# the returned variables are arrays were each cell contains
# the information of the identified blocks
[blocks,id_lin,id_col] =
id_blocks (AN_mxm,p,q,r,s,plucker_mxm,wrench_mxm);
[l_block,c_block]=size(blocks);
# This code separates each block and its identification
# vectors from the array that contains them, it also obtains
# the number of primary variables 'pv' of the block.
# primary variables = number of dependent columns
 for i=1:c_block
```

```
 printf('\n')
 printf('========================================= \n')
 printf('+++++++++++++++++++++++++++++++++++++++++ \n')
 printf('========================================= \n')
 identified_block_i = i
 printf('========================================= \n')
 printf('+++++++++++++++++++++++++++++++++++++++++ \n')
 printf('========================================= \n')
 bi=blocks{1,i}
 id_lin_i=id_lin{1,i}
 id_col_i=id_col{1,i}
 t_rank = rank(bi)
 n_c_i = columns(bi);
 pv_i = n_c_i - t_rank
 # if the primary variables are zero it means that all the
 # columns in the block are independent from each other.
 if pv_i==0
  printf('\n')
  printf('+++++++++++++++++++++++++++++++++ \n')
  printf('  All Independent columns :  \n')
  printf('+++++++++++++++++++++++++++++++++ \n')
  c_1 = bi;
  id_1 = id_col_i;
  c_0 = [ ];
  id_0 = [ ];
  c_indep = bi;
####################
  dep{1,i} = [ ];
  id_dep{1,i} = [ ];
  indep{1,i} = bi;
  id_indep{1,i} = id_col_i;
####################
 else
# the block has one or more primary variables, this code
# calls a function to find columns of zeros that justify
# the number of primary variables.
# the founded columns of zero are considered dependent and
# those non zero columns are considered independent.
  clear [c_0 id_0 nc_0 c_1 id_1 nc_1];
  bi;
  [c_0, id_0, nc_0, c_1, id_1, nc_1] =
  find_col_zeros (bi,id_col_i);
 nc_0
  if nc_0 == pv_i
   printf('\n')
```

```
   printf('******************************** \n');
   printf(' ALL Dependent columns are zero: \n');
   printf('******************************** \n');
   c_dep = c_0;
   nc_dep = columns(c_1);
   id_dep_0 = id_0;
   printf('++++++++++++++++++++++++++++++++ \n');
   printf('  ALL Independent columns found:  \n');
   printf('++++++++++++++++++++++++++++++++ \n');
   c_indep = c_1;
   nc_indep = columns (c_1);
####################
  dep{1,i} = c_0;
  id_dep{1,i} = id_0;
  indep{1,i} = c1;
  id_indep{1,i} = id_1;
####################
  elseif nc_0 == 0
   printf('\n')
   printf('------------------------ \n');
   printf(' NO zero columns found:  \n');
   printf('------------------------ \n');
   c_0 = [ ];
   id_0 = [ ];
   nc_0=0;
   pv_i_r = pv_i;
#   printf('-----------------------------------\n');
#   printf(' columns are non-all zero matrix:  \n');
#   printf('-----------------------------------\n');
   nc_1;
   c_1;
   id_1;
   [c_1_indep,id_1_indep,c_1_dep,id_1_dep]=enc(c_1,id_1,pv_i_r);
   printf('------------------ \n');
   printf('Dependent sets:    \n');
c_1_dep;
id_1_dep;
num_dep = columns (c_1_dep)
   printf('------------------ \n');
   printf(' Independent sets: \n');
   c_1_indep;
   id_1_indep;
   num_indep = columns (c_1_indep)
####################
  dep{1,i} = c_1_dep;
```

```
  id_dep{1,i} = id_1_dep;
  indep{1,i} = c_1_indep;
  id_indep{1,i} = id_1_indep;
####################
# the number of columns of zero did not justify all the
# dependencies in the block, the remaining primary variables
#
  else
   printf('\n')
   printf('------------------------------\n');
   printf(' dependent zero columns found:  \n');
   printf('------------------------------\n');
   c_dep {1,1} = c_0;
   nc_dep = columns(c_0);
   id_dep {1,1} = id_0;
   nc_0;
   pv_i_r = pv_i-nc_0;
   printf('------------------------------\n');
   printf('remaining non-all zero matrix:\n');
   printf('------------------------------\n');
   nc_1;
   c_1;
   id_1;
   [c_1_indep,id_1_indep,c_1_dep,id_1_dep]=enc(c_1,id_1,pv_i_r);
   printf('******************************** \n');
   printf(' Dependencies found in matrix \n');
   printf('******************************** \n');
c_dep {1,2} = c_1_dep;
c_dep {1,2} = id_1_dep;
   printf('++++++++++++++++++++++++++++++++ \n');
   printf(' Independent found in matrix \n');
   printf('++++++++++++++++++++++++++++++++ \n');
   c_1_indep;
   id_1_indep;
####################
  dep{1,i} = c_dep;
  id_dep{1,i} = id_dep;
  indep{1,i} = c_1_indep;
  id_indep{1,i} = id_1_indep;
####################
  endif
 endif
# indep {1,i} = c_1;
# id_indep {1,i} = id_1;
#
```

```
# dep {1,i} = c_0;
# id_dep {1,i} = id_0;
 endfor
   printf('\n')
   printf('=_=_=_=_=_=_=_=_=_=_=_=_=_=_=_=_=_=_=_=_=_=_=_=_= \n');
   printf('\n')
   printf('Set of Independent columns in each block :\n');
   printf('\n')
   printf('=_=_=_=_=_=_=_=_=_=_=_=_=_=_=_=_=_=_=_=_=_=_=_=_= \n');
 indep;
 id_indep
   printf('\n')
   printf('=_=_=_=_=_=_=_=_=_=_=_=_=_=_=_=_=_=_=_=_=_=_=_=_= \n');
   printf('\n')
   printf('Set of dependent columns in each block :\n');
   printf('\n')
   printf('=_=_=_=_=_=_=_=_=_=_=_=_=_=_=_=_=_=_=_=_=_=_=_=_= \n');
 dep;
 id_dep
endif
endfunction
```

APPENDIX K – find_col_zeros.m

```
function [c_indep,id_indep,c_dep,id_dep] = enc (N,id_col,vpr)
# Written by: Victor Carreto - 2010
# Inputs:
# N - matrix
# id_col - column array identificating the columns of N
# Outputs:
# c_indep - column array of sets of independent columns,
# id_indep - columns array identificating the indep. columns,
# c_dep - column array of sets of dependent columns,
# id_dep - columns array identificating the dep. columns.
#
# determine the size of the matrix,
[ l_N, c_N ] = size ( N );
# creates an internal identification vector
aux_id = 1:c_N;
N;
rank_N = rank(N);
id_col;
vpr
count_dep = 0;
count_indep = 0;
# If there are non primary variables then all the columns in
# the matrix are independent.
if vpr == 0
 printf ("---------------------------------\n")
 printf (" There are ZERO primary variables \n")
 printf ("       all the columns are       \n")
 printf ("           INDEPENDENT           \n")
 printf ("---------------------------------\n")
 c_indep=N;
 id_indep=id_col;
 c_dep=[];
 id_dep=[];
# If there is one or more primary variables, the following
# code will identify sets of dependent columns that justify
# the number of primary variables.
elseif vpr >= 1
# The function combinacion.m returns a combinatorial matrix
# indicates which columns are going to be removed to see if
# each line they are independent or not.
 for j=1:2
  [ pc ] = combinacion (N, j);
  [l_pc, c_pc ] = size ( pc );
 for i = 1 : l_pc
#  printf (" \n")
```

```
#  printf ("***************************** \n")
#  printf ("****  NUEVA COMBINACION  **** \n")
#  printf ("***************************** \n")
  N;
  comb_i = pc (i,:);
  c_e = N (:,comb_i);
  id_c_e = id_col (1,comb_i);
  c_n_e = N;
  c_n_e (:,comb_i) = [ ];
  id_c_n_e = id_col;
  id_c_n_e (:,comb_i) = [ ];
# The non removed columns form the a matrix with a
# term rank of the input matrix the difference between the
# rank and the term-rank will indicate if the
# removed columns are indeed independent or not.
  t_rank = rank(c_n_e);
  if t_rank==rank_N
# If the rank of the input matrix is equal to the rank of
# the non removed columns matrix then the removed columns
# are linearly dependent
#   printf (" \n")
#   printf ("----------------------------  \n")
#   printf ("El rango es igual, \n ")
#   printf ("la combinacion es DEPENDIENTE  \n")
#   printf ("----------------------------  \n")
   count_dep = count_dep + 1;
   c_dep{1,count_dep} = c_e;
   id_dep{1,count_dep} = id_c_e;
  else
#   printf (" \n")
#   printf ("------------------------------  \n")
#   printf ("El rango es igual, \n ")
#   printf ("la combinacion es INDEPENDIENTE  \n")
#   printf ("------------------------------  \n")
   count_indep = count_indep + 1;
   c_indep {1,count_indep} = c_e;
   id_indep {1,count_indep} = id_c_e;
#  The diference between the ranks implies that
#  the removed columns are independent.
  endif
 endfor
 endfor
endif
   count_dep;
   count_indep;
```

```
    c_indep;
    id_indep;
    c_dep;
    id_dep;
endfunction
```

APPENDIX L – enc.m

```
function [c_indep,id_indep,c_dep,id_dep] = enc (N,id_col,vpr)
# Written by: Victor Carreto - 2010
# Inputs:
# N - matrix
# id_col - column array identificating the columns of N
# Outputs:
# c_indep - column array of sets of independent columns,
# id_indep - columns array identificating the indep. columns,
# c_dep - column array of sets of dependent columns,
# id_dep - columns array identificating the dep. columns.
#
# determine the size of the matrix,
[ l_N, c_N ] = size ( N );
# creates an internal identification vector
aux_id = 1:c_N;
N;
rank_N = rank(N);
id_col;
vpr
count_dep = 0;
count_indep = 0;
# If there are non primary variables then all the columns in
# the matrix are independent.
if vpr == 0
 printf ("---------------------------------\n")
 printf (" There are ZERO pimary variables \n")
 printf ("       all the columns are       \n")
 printf ("           INDEPENDENT           \n")
 printf ("---------------------------------\n")
 c_indep=N;
 id_indep=id_col;
 c_dep=[];
 id_dep=[];
# If there is one or more primary variables, the following
# code will identify sets of dependent columns that justify
# the number of primary variables.
elseif vpr >= 1
# The function combinacion.m returns a combinatorial matrix
# indicates which columns are going to be removed to see if
# each line they are independent or not.
 for j=1:2
  [ pc ] = combinacion (N, j);
  [l_pc, c_pc ] = size ( pc );
 for i = 1 : l_pc
#  printf (" \n")
```

```
#  printf ("***************************** \n")
#  printf ("****  NEW COMBINATION    **** \n")
#  printf ("***************************** \n")
  N;
  comb_i = pc (i,:);
  c_e = N (:,comb_i);
  id_c_e = id_col (1,comb_i);
  c_n_e = N;
  c_n_e (:,comb_i) = [ ];
  id_c_n_e = id_col;
  id_c_n_e (:,comb_i) = [ ];
# The non removed columns form the a matrix with a
# term rank of the input matrix the difference between
# the rank and the term-rank will indicate if the
# removed columns are indeed independent or not.
  t_rank = rank(c_n_e);
  if t_rank==rank_N
# If the rank of the input matrix is equal to the rank of
# the non removed columns matrix then the removed columns
# are linearly dependent
#   printf (" \n")
#   printf ("----------------------------  \n")
#   printf ("The rank is the same, \n ")
#   printf ("the combination is dependent  \n")
#   printf ("----------------------------  \n")
   count_dep = count_dep + 1;
   c_dep{1,count_dep} = c_e;
   id_dep{1,count_dep} = id_c_e;
  else
#   printf (" \n")
#   printf ("------------------------------  \n")
#   printf ("The rank is different, \n ")
#   printf ("the combination is independent  \n")
#   printf ("------------------------------  \n")
   count_indep = count_indep + 1;
   c_indep {1,count_indep} = c_e;
   id_indep {1,count_indep} = id_c_e;
#  The diference between the ranks implies that
#  the removed columns are independent.
  endif
 endfor
 endfor
endif
   count_dep;
   count_indep;
```

```
    c_indep;
    id_indep;
    c_dep;
    id_dep;
endfunction
```

APPENDIX M – ex_1.m

```
function ex_1
printf (" \n");
printf (" MECHANISM :                                    \n");
printf (" \n");
printf ("     y ^                                        \n");
printf ("       |    b                                   \n");
printf ("       |    o                                   \n");
printf ("       |   /  ‘                                 \n");
printf ("       |1 /     ‘    2                          \n");
printf ("       | /        ‘                             \n");
printf ("       |/           ‘ ___ 3                     \n");
printf ("       o -----------| o  | -------------> x     \n");
printf ("     a         0    |____| d                    \n");
printf ("                       c                        \n");
printf (" \n");
printf (" GRAPH ’Gc’                                     \n");
printf ("              b                                 \n");
printf ("  1 x--------------->----x 2                    \n");
printf ("    |                    |                      \n");
printf ("    ^                    |                      \n");
printf ("  a |                    | c                    \n");
printf ("    |                    v                      \n");
printf ("    |                    |                      \n");
printf ("  0 x================>===>x 3                    \n");
printf ("              d                                 \n");
printf (" \n");
############################################################
############################################################
############################################################
printf (" \n");
printf ("********************************** \n");
printf ("***********   CASE I   *********** \n");
printf ("**********   FREEDOMS   ********** \n");
printf ("********************************** \n");
printf (" \n");
printf (" a (a1)= rotation  in ’z’ axis\n");
printf (" b (b1)= rotation  in ’z’ axis\n");
printf (" c (c1)= rotation  in ’z’ axis\n");
printf (" d (d1)= translation in ’x’ axis\n");
printf (" \n");
#
# Motion Analysis
#         a  b  c  d
#         a1 b1 c1 d1
h_m =  [  0  0  0  1  ]
```

```
Sm =   [  0  0  0  1  ;
          0  0  0  0  ;
          1  1  1  0  ]
S0_m = [  0  2  6  6  ;
          0  2  0  0  ;
          0  0  0  0  ]
BM =   [  1  1  1 -1  ]
kinematic_analysis (h_m, Sm, S0_m, BM)
############################################################
printf (" \n");
printf ("********************************** \n");
printf ("***********  CASE I   *********** \n");
printf ("*******    CONSTRAINTS    ******** \n");
printf ("********************************** \n");
printf (" \n");
printf (" a (a1)= translation  in 'x' axis\n");
printf (" b (a2)= translation  in 'y' axis\n");
printf (" c (a3)= translation  in 'z' axis\n");
printf (" d (a4)= rotation  in 'x' axis\n");
printf (" e (a5)= rotation  in 'y' axis\n");
printf (" f (b1)= translation  in 'x' axis\n");
printf (" g (b2)= translation  in 'y' axis\n");
printf (" h (b3)= translation  in 'z' axis\n");
printf (" i (b4)= rotation  in 'x' axis\n");
printf (" j (b5)= rotation  in 'y' axis\n");
printf (" k (c1)= translation  in 'x' axis\n");
printf (" l (c2)= translation  in 'y' axis\n");
printf (" m (c3)= translation  in 'z' axis\n");
printf (" n (c4)= rotation  in 'x' axis\n");
printf (" o (c5)= rotation  in 'y' axis\n");
printf (" p (d1)= translation  in 'y' axis\n");
printf (" q (d2)= translation  in 'z' axis\n");
printf (" r (d3)= rotation  in 'x' axis\n");
printf (" s (d4)= rotation  in 'y' axis\n");
printf (" t (d5)= rotation  in 'z' axis\n");
#
# Static Analysis
#         a  b  c  d  e  f  g  h  i  j  k  l  m  n  o  p  q  r  s  t
#         a1 a2 a3 a4 a5 b1 b2 b3 b4 b5 c1 c2 c3 c4 c5 d1 d2 d3 d4 d5
h_a =  [  1  1  1  0  0  1  1  1  0  0  1  1  1  0  0  1  1  0  0  0 ]
Sa  =  [  1  0  0  1  0  1  0  0  1  0  1  0  0  1  0  0  0  1  0  0 ;
          0  1  0  0  1  0  1  0  0  1  0  1  0  0  1  1  0  0  1  0 ;
          0  0  1  0  0  0  0  1  0  0  0  0  1  0  0  0  1  0  0  1 ]
S0_a = [  0  0  0  0  0  2  2  2  2  2  6  6  6  6  6  6  6  6  6  6 ;
          0  0  0  0  0  2  2  2  2  2  0  0  0  0  0  0  0  0  0  0 ;
```

```
          0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 ]
QA =   [  1  1  1  1  1  0  0  0  0  0  0  0  0  0  0  1  1  1  1  1 ;
          0  0  0  0  0  1  1  1  1  1  0  0  0  0  0  1  1  1  1  1 ;
          0  0  0  0  0  0  0  0  0  0  1  1  1  1  1  1  1  1  1  1 ]
static_analysis (h_a, Sa, S0_a, QA)
############################################################
############################################################
############################################################
printf (" \n");
printf ("********************************** \n");
printf ("**********   CASE II  ************ \n");
printf ("**********   FREEDOMS   ********** \n");
printf ("********************************** \n");
printf (" \n");
printf ("changing couplings 'b' and 'd' \n");
printf (" \n");
printf (" a (a1)= rotation  in 'z' axis \n");
printf (" b (b1)= rotation  in 'x' axis \n");
printf (" c (b2)= rotation  in 'y' axis \n");
printf (" d (b3)= rotation  in 'z' axis \n");
printf (" e (c1)= rotation  in 'z' axis \n");
printf (" f (d1)= rotation  in 'x' axis \n");
printf (" g (d2)= translation in 'x' axis \n");
printf (" \n");
#
# Motion Analysis
#         a  b  c  d  e  f  g
#         a1 b1 b2 b3 c1 d1 d2
h_m =  [  0  0  0  0  0  0  1  ]
Sm =   [  0  1  0  0  0  1  1   ;
          0  0  1  0  0  0  0   ;
          1  0  0  1  1  0  0   ]
S0_m = [  0  2  2  2  6  6  6   ;
          0  2  2  2  0  0  0   ;
          0  0  0  0  0  0  0   ]
BM  =  [  1  1  1  1  1 -1 -1   ]
kinematic_analysis (h_m, Sm, S0_m, BM)
############################################################
printf (" \n");
printf ("********************************** \n");
printf ("***********  CASE II   *********** \n");
printf ("*******    CONSTRAINTS    ******** \n");
printf ("********************************** \n");
printf (" \n");
printf (" a (a1)= translation  in 'x' axis\n");
```

```
printf (" b (a2)= translation  in 'y' axis\n");
printf (" c (a3)= translation  in 'z' axis\n");
printf (" d (a4)= rotation  in 'x' axis\n");
printf (" e (a5)= rotation  in 'y' axis\n");
printf (" f (b1)= translation  in 'x' axis\n");
printf (" g (b2)= translation  in 'y' axis\n");
printf (" h (b3)= translation  in 'z' axis\n");
printf (" i (c1)= translation  in 'x' axis\n");
printf (" j (c2)= translation  in 'y' axis\n");
printf (" k (c3)= translation  in 'z' axis\n");
printf (" l (c4)= rotation  in 'x' axis\n");
printf (" m (c5)= rotation  in 'y' axis\n");
printf (" n (d1)= translation  in 'y' axis\n");
printf (" o (d2)= translation  in 'z' axis\n");
printf (" p (d3)= rotation  in 'y' axis\n");
printf (" q (d4)= rotation  in 'z' axis\n");
#
# Static Analysis
#       a  b  c  d  e  f  g  h  i  j  k  l  m  n  o  p  q
#       a1 a2 a3 a4 a5 b1 b2 b3 c1 c2 c3 c4 c5 d1 d2 d4 d5
h_a = [ 1  1  1  0  0  1  1  1  1  1  1  0  0  1  1  0  0 ]
Sa  = [ 1  0  0  1  0  1  0  0  1  0  0  1  0  0  0  0  0 ;
        0  1  0  0  1  0  1  0  0  1  0  0  1  1  0  1  0 ;
        0  0  1  0  0  0  0  1  0  0  1  0  0  0  1  0  1 ]
S0_a= [ 0  0  0  0  0  2  2  2  6  6  6  6  6  6  6  6  6 ;
        0  0  0  0  0  2  2  2  0  0  0  0  0  0  0  0  0 ;
        0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 ]
QA  = [ 1  1  1  1  1  0  0  0  0  0  0  0  0  1  1  1  1 ;
        0  0  0  0  0  1  1  1  0  0  0  0  0  1  1  1  1 ;
        0  0  0  0  0  0  0  0  1  1  1  1  1  1  1  1  1 ]
static_analysis (h_a, Sa, S0_a, QA)
##########################################################
##########################################################
##########################################################
endfunction
```